Engineer

Bash

GNU-skallen, de fleste scripts er skrevet til

Hvad det er

Bash, kort for Bourne-Again SHell, distribueret som GNU Bash og startet med kommandoen `bash`, er et program, der læser kommandoer, skrevet ved en prompt eller gemt i en fil, og udfører dem: det starter andre programmer, kobler deres input og output sammen og tilføjer variabler, løkker, tests og funktioner, så en række kommandoer kan træffe beslutninger. Navnet er et ordspil på Stephen Bourne, hvis `sh` fulgte med den syvende udgave af Research Unix fra Bell Labs. Manualen beskriver Bash som stort set kompatibel med den skal, med funktioner hentet fra Korn-skallen og C-skallen, og som tænkt til at følge POSIX-standardens del om skallen. Den aktuelle udgave er 5.3, hvis manual er dateret maj 2025.

En skal er noget andet end det vindue, den kører i: terminalen tegner tegn og sender tastetryk videre, og skallen er programmet bagved, der fortolker dem. Bash kører interaktivt, med prompt, historik og tab-fuldførelse, eller ikke-interaktivt, hvor den læser et script fra top til bund uden opsyn, og det er dér, de fleste af dens fælder ligger.

Bash har konkurrenter i samme rolle. POSIX definerer et portabelt skalsprog, `sh`, og hvilket program der ligger som `/bin/sh`, afhænger af systemet. På Debian og Ubuntu er det dash, en lille POSIX-skal, der nedstammer fra Almquist-skallen, og Debian Policy nævner bash, dash og ksh som de mest almindelige skaller at bruge som `/bin/sh`. Zsh har været standardskal for nye konti på macOS siden version 10.15, selvom Mac'en stadig har en `/bin/bash`. Et script vælger sin fortolker med første linje, shebang-linjen: `#!/bin/sh` beder om den POSIX-skal, systemet nu engang har, mens `#!/bin/bash` eller `#!/usr/bin/env bash` beder om Bash ved navn.

Brug begrebet i en beslutning

Brug »Hvad er Bash?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Daglig drift af servere, mail, konti og opdateringer, så det, alle er afhængige af, forbliver kedeligt.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har drevet 10 PostgreSQL- og MS SQL-servere — 10 databaseservere udrullet og holdt i god drift

Har drevet 10 PostgreSQL- og MS SQL-servere Kontakt os →

Hvorfor det betyder noget

Bash er det sprog, der binder Unix-servere sammen. Den findes på næsten alle Linux-maskiner, og de kommandoer, en ingeniør skriver ved prompten, er allerede Bash, så en håndfuld af dem gemt i en fil er et script uden nyt sprog, ny pakke eller runtime at holde opdateret. Til at kæde eksisterende programmer sammen, for eksempel at dumpe en database, komprimere dumpet og kopiere det et andet sted hen, er få sprog så kortfattede, fordi pipes, omdirigering og exitstatus er dens modersmål.

Netop de egenskaber bliver til omkostninger, når et script vokser. Alle værdier er tekststrenge, og aritmetik regner kun med heltal, så alt med struktur, som JSON, datoer eller beløb med decimaler, må gives videre til et andet værktøj. Udvidelser uden anførselstegn deles op i ord og matches derefter mod filnavne, så en værdi med et mellemrum eller en stjerne ændrer betydningen af den kommando, den havner i. Fejlhåndtering skal man selv slå til: som udgangspunkt kører et script videre, efter at en kommando er fejlet, og de indstillinger, der ændrer det, `set -e` og `pipefail`, har undtagelser, der fylder flere afsnit i manualen. En brugbar tommelfingerregel er, at når et script har brug for datastrukturer, en parser eller en testsuite, er det blevet et program, og så er et sprog som Python billigere at vedligeholde.

Portabilitet er også et valg. Debian Policy kræver, at `/bin/sh`-scripts holder sig til POSIX-funktioner og en kort liste af udvidelser, og at et script, der har brug for mere, angiver sin skal i første linje, for eksempel `#!/bin/bash`. Den foreslår `checkbashisms` eller en kørsel under en strengere skal som posh til at finde overtrædelserne og anbefaler `/bin/bash`, når man er i tvivl. Standarden flytter sig langsomt: Bash har haft `pipefail` siden version 3.0, og POSIX tog den først med i 2024-udgaven.

En skal gør tekst til kommandoer, og det gør den til en sikkerhedsgrænse alle steder, hvor tekst udefra når frem til den. Shellshock, registreret som CVE-2014-6271 i september 2014, viste, hvor bred den grænse kan være. Bash til og med version 4.3 udførte kommandoer, der stod efter en funktionsdefinition i en miljøvariabel, så alt, der kopierede en angribers tekst ind i miljøet, før Bash blev startet, blev en vej til at køre kode på maskinen, blandt andet CGI-scripts under Apache, ForceCommand-funktionen i OpenSSH og visse DHCP-klienter. Den første rettelse var ufuldstændig, og det, der stod tilbage, fik sit eget nummer, CVE-2014-7169. Den daglige lære er mere jordnær: tekst udefra skal nå en kommando som et argument og aldrig som en del af en streng, der gives til `eval` eller `sh -c`.

Sådan ser det ud i praksis

I virksomhedens arbejde har Bash den plads, teknologisiden giver den: Python tager sig af dataarbejde, automatisering og tjenester, og Bash er limen imellem, skrevet til at blive læst.

I de seks ETL- og ELT-pipelines, der forsynede Google BigQuery med data fra et data warehouse bag en IPsec VPN, indeholdt de hentede arkiver Parquet-, CSV- og .bak-filer. Scripts i Bash og Python trak filerne ud og sorterede hver enkelt efter type og skema, før bcp, psql og SSIS lagde dem i staging-skemaer, og gzip, Google Cloud Storage og bq bragte resultaterne videre til BigQuery. Automatiseringen skar mindst tre timers manuelt arbejde ned til under 20 minutter for hele forløbet og flyttede data fra ugentlig til daglig synkronisering.

De hundrede automatiserede kritiske databackups viser den samme arbejdsdeling. Barman tog sig af PostgreSQL-siden, Google Cloud holdt kopierne uden for huset, og Bash og Python styrede og validerede hele ordningen, med opbevaringsregler, der holdt kopierne aktuelle, og integritetstjek, der bekræftede, at hver enkelt var i orden og mere end blot til stede. Det er den slags opgave, Bash passer til: en række eksisterende værktøjer, hvor scriptets værdi ligger i at tjekke, hvad hvert trin returnerede.

Hvor det går galt

Et filnavn med mellemrum bliver til to argumenter. En `$file` uden anførselstegn deles ved blanktegn og udvides derefter som et glob-mønster, så `report 2024.csv` når frem til `rm` som to navne, og en vildfaren `*` rammer en

hel mappe. Sæt anførselstegn om hver udvidelse, som i `"$file"`, giv argumentlister videre som `"$@"`, hold lister af navne i arrays, og lad ShellCheck finde resten.

En backup-pipeline melder succes, selvom dumpet er fejlet. En pipelines exitstatus er den sidste kommandos, så i `pg_dump app | gzip > app.sql.gz` lykkes gzip stadig med det lille, den nåede at modtage, når dumpet fejler. Slå `set -o pipefail` til, så pipelinen returnerer den sidste status, der er forskellig fra nul, og kontrollér resultatet med `gzip -t` eller helst med en prøvegendannelse.

Et script med `set -e` kører videre forbi en fejlet kommando. Indstillingen ignoreres i testen i en `if` eller `while`, i alle dele af en `&&`- eller `||`-liste undtagen den sidste og i enhver funktion, der kaldes fra de steder, så dumpet nedenfor kan fejle, mens scriptet alligevel skriver sin succesbesked:

```
#!/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 ignoreres her inde i backup
  echo "backup done"
fi
```

En beslægtet fælde er `local out=$(cmd)`, som returnerer status for `local` og skjuler en fejl i `cmd`. Følg i funktioner hver vigtig kommando med `|| return 1`, og erklær en variabel på én linje og tildel den en værdi på den næste.

Et script, der virker på én maskine, går i stykker på en anden. En fil, der starter med `#!/bin/sh`, men bruger `[[`-tests eller arrays, kører under Bash på ét system og under dash på det næste. Dér er `[[` en ukendt kommando, så en `if`, der bygger på den, skriver en fejl og går til sin else-gren, og en array-tildeling er en syntaksfejl, der afslutter scriptet. På en Mac melder `/bin/bash` version 3.2.57, som er ældre end de associative arrays og `mapfile`, der kom i 4.0, så `declare -A` stopper dér med en fejl om en ugyldig option. Angiv den skal, scriptet reelt har brug for, i shebang-linjen, og test mod den ældste version, det skal køre på.

En oprydningslinje sletter langt mere, end den skulle. Er `build_dir` ikke sat, udvides `rm -rf "$build_dir"/*` til `rm -rf /*`. `set -u` gør en variabel, der ikke er sat, til en fejl, og skriver man `"${build_dir:?}"`, stopper et ikke-interaktivt script, hver gang variablen mangler eller indeholder en tom streng.

Læs mere

- GNU Bash Reference Manual — manualen til 5.3: udvidelser, citering, den indbyggede kommando `set` og de præcise tilfælde, hvor `set -e` ignoreres.
- Shell Command Language — det portable `sh`-sprog i POSIX.1-2024 og dermed grænsen for ethvert `#!/bin/sh`-script.
- Debian Policy: Scripts — hvad `/bin/sh` skal understøtte på Debian, hvornår et script skal bede om Bash ved navn, og rådene om `set -e`.
- Bash NEWS — vedligeholderens liste over nye funktioner udgave for udgave, til at tjekke, hvilken version en konstruktion kræver.
- CVE-2014-6271 — NVD's post om Shellshock med de angrebsveje, den nævner, og nummeret på opfølgeren.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er Bash?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/bash/