Engineer

Bash

>_ GNU-skallet de fleste skript skrives for

Hva det er

Bash, Bourne-Again SHell, distribuert som GNU Bash og startet med kommandoen `bash`, er et program som leser kommandoer, skrevet inn ved en ledetekst eller lagret i en fil, og utfører dem: det starter andre programmer, kobler sammen inn- og utdataene deres, og legger til variabler, løkker, tester og funksjoner, slik at en rekke kommandoer kan ta avgjørelser. Navnet er et ordspill på Stephen Bourne, som skrev `sh`, skallet som fulgte med den sjuende utgaven av Research Unix fra Bell Labs. Manualen beskriver Bash som i stor grad kompatibelt med det skallet, med funksjoner hentet fra Korn-skallet og C-skallet, og som ment å følge skalldelen av POSIX-standarden. Den gjeldende utgaven er 5.3, og manualen for 5.3 er datert mai 2025.

Et skall er lett å forveksle med vinduet det kjører i: terminalen tegner tegn og sender tastetrykk videre, og skallet er programmet på den andre siden som tolker dem. Bash kjører interaktivt, med ledetekst, historikk og fullføring med tab-tasten, eller ikke-interaktivt, der det leser et skript fra topp til bunn uten at noen følger med, og det er der de fleste fellene ligger.

Bash er ett av flere programmer som kan fylle den samme rollen. POSIX definerer et portabelt skallspråk, `sh`, og programmet som er installert som `/bin/sh`, varierer fra system til system. På Debian og Ubuntu er det dash, et lite POSIX-skall som stammer fra Almquist-skallet, og Debian Policy nevner bash, dash og ksh som de vanligste skallene å bruke der. Zsh har vært standard innloggingsskall for nye kontoer på macOS siden versjon 10.15, selv om Mac-en fortsatt har en `/bin/bash`. Et skript velger tolken sin med første linje, shebang-linjen: `#!/bin/sh` ber om det POSIX-skallet systemet har, mens `#!/bin/bash` eller `#!/usr/bin/env bash` ber om Bash ved navn.

Bruk begrepet i en beslutning

Bruk «Hva er Bash?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Daglig ettersyn av servere, e-post, kontoer og oppdateringer, slik at det alle er avhengige av, forblir kjedelig.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Drev 10 PostgreSQL- og MS SQL-servere på Ubuntu — 10 databaseservere produksjonssatt og holdt i god stand

Drev 10 PostgreSQL- og MS SQL-servere på Ubuntu Ta kontakt →

Hvorfor det er viktig

Bash er limspråket på Unix-servere. Det finnes på nesten hver Linux-maskin, og kommandoene en utvikler skriver ved ledeteksten, er allerede Bash, så en håndfull av dem lagret i en fil blir et skript som kjører på det maskinen allerede har. Til å kjede sammen eksisterende programmer, f.eks. å dumpe en database, komprimere dumpen og kopiere den et annet sted, er få språk like korte, fordi pipes, omdirigering og avslutningsstatus er grunnordforrådet i språket.

De samme egenskapene blir til kostnader når et skript vokser. Hver verdi er en streng, og aritmetikk virker bare på heltall, så alt med struktur, som JSON, datoer eller beløp med desimaler, må overlates til et annet verktøy. Ekspansjoner uten anførselstegn deles opp i ord og matches deretter mot filnavn, så en verdi som inneholder et mellomrom eller en stjerne, endrer betydningen av kommandoen den havner i. Feilhåndtering må slås på: som standard fortsetter et skript etter at en kommando har feilet, og valgene som endrer det, `set -e` og `pipefail`, kommer med unntak manualen trenger flere avsnitt for å ramse opp. En regel som fungerer, er at når et skript trenger datastrukturer, en parser eller en testsuite, har det blitt et program, og et språk som Python er billigere å vedlikeholde.

Portabilitet er også en beslutning. Debian Policy krever at `/bin/sh`-skript holder seg til POSIX-funksjoner pluss en kort liste med utvidelser, og ber hvert skript som trenger mer, om å oppgi skallet sitt på første linje, f.eks. `#!/bin/bash`. Den foreslår `checkbashisms` eller en kjøring under et strengere skall som posh for å finne brudd, og råder til `/bin/bash` i tvilstilfeller. Standarden beveger seg sakte: Bash har tilbudt `pipefail` siden versjon 3.0, og POSIX tok det inn først i 2024-utgaven.

Et skall gjør tekst om til kommandoer, og det gjør det til en sikkerhetsgrense overalt der tekst utenfra når det. Shellshock, registrert som CVE-2014-6271 i september 2014, viste hvor bred den grensen kan være. Bash til og med versjon 4.3 kjørte kommandoer som fulgte etter en funksjonsdefinisjon i en miljøvariabel, så alt som kopierte tekst en angriper styrte, inn i miljøet før Bash ble startet, deriblant CGI-skript under Apache, ForceCommand-funksjonen i OpenSSH og noen DHCP-klienter, ble en vei til å kjøre kode på maskinen. Den første rettelsen var ufullstendig, og det som gjensto, fikk sin egen identifikator, CVE-2014-7169. Den daglige lærdommen er mindre: tekst utenfra skal nå en kommando som et argument, aldri som en del av en streng som gis til `eval` eller `sh -c` .

Slik ser det ut i praksis

I selskapets arbeid har Bash den jobben siden om teknologiene gir det: Python til dataarbeid, automatisering og tjenester, og Bash til limet, skrevet for å bli lest.

I de seks ETL- og ELT-pipelinene som fylte Google BigQuery fra et datavarehus bak et IPsec-VPN, inneholdt de nedlastede arkivene Parquet-, CSV- og .bak-filer. Skript i Bash og Python pakket dem ut og sorterte hver fil etter type og struktur før bcp, psql og SSIS lastet dem inn i staging-skjemaer, og gzip, Google Cloud Storage og verktøyet bq fraktet resultatene videre til BigQuery. Da utpakkingen og sorteringen var skriptet, ble en kjøring som hadde tatt minst tre timer for hånd, ferdig på under 20 minutter, og dataene kunne synkroniseres hver dag i stedet for én gang i uka.

De hundre automatiserte kritiske sikkerhetskopiene av data viser den samme arbeidsdelingen. Barman sto for PostgreSQL-delen, kopiene utenfor huset lå i Google Cloud, og Bash og Python styrte og kontrollerte hele ordningen: regler for oppbevaring holdt kopiene ferske, og integritetskontroller viste at hver kopi faktisk var brukbar, og ikke bare at den fantes. Det er jobben Bash passer til: en rekke eksisterende verktøy, der verdien i skriptet ligger i å sjekke hva hvert steg returnerte.

Hvor det går galt

Et filnavn med mellomrom blir to argumenter. En `$file` uten anførselstegn deles ved mellomrom og ekspanderes deretter som et glob-mønster, så `report 2024.csv` når `rm` som to navn, og en løs `*` treffer en hel katalog. Sett

anførselstegn rundt hver ekspansjon, som i `"$file"`, send argumentlister videre som `"$@"`, hold lister med navn i arrayer, og la ShellCheck finne resten.

En pipeline for sikkerhetskopiering melder suksess etter at dumpen har feilet. Avslutningsstatusen til en pipeline er statusen til den siste kommandoen, så i `pg_dump app | gzip > app.sql.gz` lykkes gzip fortsatt med det lille den fikk, selv om dumpen feilet. Slå på `set -o pipefail`, som får pipelinen til å returnere den siste statusen som ikke er null, og sjekk resultatet med `gzip -t` eller, enda bedre, en testgjenoppretting.

Et skript med `set -e` fortsetter forbi en kommando som feilet. Valget ignoreres i testen i en `if` eller `while`, i hver del av en `&&`- eller `||`-liste unntatt den siste, og inne i hver funksjon som kalles fra de stedene, så dumpen nedenfor kan feile mens skriptet likevel skriver ut suksessmeldingen sin:

```
#!/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
```

En beslektet felle er `local out=$(cmd)`, som returnerer statusen til `local` og skjuler en feil i `cmd`. Inne i funksjoner bør hver kommando som betyr noe, følges av `|| return 1`, og en variabel bør deklareres på en egen linje før den får en verdi.

Et skript som virker på én maskin, går i stykker på en annen. En fil som starter med `#!/bin/sh`, men bruker `[[`-tester eller arrayer, kjører under Bash på ett system og under dash på det neste. Der er `[[` en ukjent kommando, så en `if` bygd på den skriver ut en feil og velger else-grenen, og en tilordning til en array er en syntaksfeil som avslutter skriptet. På en Mac melder `/bin/bash` versjon 3.2.57, eldre enn de assosiative arrayene og `mapfile` som kom i 4.0, så `declare -A` stopper der med en feil om ugyldig valg. Oppgi skallet skriptet faktisk trenger, i shebang-linjen, og test mot den eldste versjonen det må kjøre på.

En oppryddingslinje sletter langt mer enn den skulle. Når `build_dir` ikke er satt, ekspanderer `rm -rf "$build_dir"/*` til `rm -rf /*`. `set -u` gjør en variabel som ikke er satt, til en feil, og å skrive `"${build_dir:?}"` stopper et ikke-interaktivt skript når variabelen ikke er satt eller er satt til en tom streng.

Les mer

- GNU Bash Reference Manual — manualen for 5.3: ekspansjoner, anførselstegn, den innebygde kommandoen `set` og nøyaktig når `set -e` blir ignorert.
- Shell Command Language — det portable `sh`-språket i POSIX.1-2024, grensen for ethvert `#!/bin/sh`-skript.
- Debian Policy: Scripts — hva `/bin/sh` må støtte på Debian, når et skript må be om Bash ved navn, og rådene om `set -e`.
- Bash NEWS — vedlikeholderens liste over nye funksjoner, utgave for utgave, for å se hvilken versjon en konstruksjon krever.
- CVE-2014-6271 — NVD-oppføringen for Shellshock, med angrepsveiene den lister opp og oppfølgingsidentifikatoren.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er Bash?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/bash/