Ghid Practic de Automatizare cu Bash

Dezvoltarea și depanarea script-urilor shell (timp lucru – 120 minute)

Introducere în automatizare

Automatizarea sarcinilor repetitive reprezintă o componentă esențială în administrarea sistemelor moderne. Acest ghid explorează tehnicile fundamentale de creare și optimizare a script-urilor Bash, oferind metode practice pentru eficientizarea workflow-ului zilnic.

1. Fundamente de scripting

1.1 Structura de bază

Un script Bash este un fișier text care conține o serie de comenzi executabile. Structura minimă necesită un shebang și permisiuni adecvate.

```
#!/bin/bash

# Afișare informații sistem

hostname

date +%Y-%m-%d

df -h | grep /dev/sda1
```

Configurare permisiuni:

```
chmod u+x automation_script.sh
```

Executare:

```
./automation_script.sh
```

1.2 Rolul shebang-ului

Linia inițială `#!/bin/bash` specifică interpretorul pentru execuție. Absența acesteia necesită invocarea explicită:

```
bash automation_script.sh
```

Această practică devine obligatorie pentru script-uri Python sau alte limbaje interpretate:

```
#!/usr/bin/python3
```

2. Documentare și mentenanță

2.1 Comentarii eficiente

Comentariile facilitează înțelegerea logicii și mențin codul accesibil pe termen lung. Orice text precedat de # este ignorat de interpretor.

```
#!/bin/bash



# Script: Backup automated daily

# Author: System Administrator

# Date: 2025-01-15



hostname

# Verificare utilizator curent pentru audit

id -un
```

3. Gestionarea variabilelor

3.1 Definire și utilizare

Variabilele stochează date reutilizabile, reducând redundanța și îmbunătățind flexibilitatea script-urilor.

```
#!/bin/bash



BACKUP_DIR="/var/backups/database"

LOG_FILE="/var/log/backup.log"



echo "Backup location: $BACKUP_DIR" >> $LOG_FILE

date >> $LOG_FILE
```

Notă importantă: Variabilele efectuează substituție de text. Shell-ul înlocuiește referințele cu valorile stocate.

3.2 Sintaxă corectă

Atenție: Spațiile în jurul operatorului de atribuire generează erori subtile.

Incorect:

```
BACKUP_DIR ="/var/backups"    # Eroare

BACKUP_DIR= "/var/backups"    # Eroare

BACKUP_DIR = "/var/backups"   # Eroare
```

Corect:

```
BACKUP_DIR="/var/backups"
```

4. Scenarii practice

4.1 Procesare batch de fișiere

Următorul exemplu procesează fișiere de log specifice:

```
#!/bin/bash


LOG_FILES="access.log error.log"


ls -lh $LOG_FILES
```

Când executat, variabila se expandează în lista de fișiere:

```
ls -lh access.log error.log
```

4.2 Concatenare și delimitare

Utilizarea acoladelor previne ambiguitatea în concatenare:

```
#!/bin/bash

PREFIX="backup"

echo $PREFIXfile        # Caută variabila 'PREFIXfile'

echo ${PREFIX}file     # Output: backupfile
```

Important: Variabilele nedefinte nu generează erori—sunt înlocuite cu șir vid (nimic , fara spatiu)

5. Context de execuție

5.1 Directorul de lucru

Esențial: Script-urile operează în directorul din care sunt invocate, nu în cel unde rezidă.

Exemplu:

```
# Locație curentă: /home/admin/reports

# Script localizat în: /opt/scripts/process.sh

/opt/scripts/process.sh
```

Comenzile din `process.sh` se vor executa în `/home/admin/reports`, nu în `/opt/scripts`.

6. Structuri de control

6.1 Bucle iterative

Structurile `for` și `while` permit procesarea repetitivă a colecțiilor de date.

Bucla FOR

Procesare directă de fișiere:

```
#!/bin/bash



for LOG in access.log error.log system.log

do

    echo "Processing: $LOG"

    wc -l $LOG

    gzip $LOG

done
```

Utilizare cu substituție de comenzi:

```
#!/bin/bash


for FILE in $(find /var/log -name '*.log' -mtime +30)
do
    echo "Archiving old log: $FILE"
    tar -czf ${FILE}.tar.gz $FILE
    rm $FILE
done
```

Notă: Sintaxa `$(comandă)` execută comanda și înlocuiește output-ul ca text.

Utilizare cu pattern matching (globbing):

```
#!/bin/bash


for REPORT in sales_*.csv
do
    echo "Validating: $REPORT"
    python3 validate_data.py $REPORT
done
```

Bucla WHILE

Procesare linie cu linie din fișiere:

```
#!/bin/bash


while IFS= read -r line
do
    echo "Configuration entry: $line"
done < /etc/app/config.txt
```

Variabila `line` conține fiecare linie succesiv până la epuizarea fișierului.

6.2 Structuri condiționale

Structura `if-else` permite execuție condiționată pe baza evaluării expresiilor.

```
#!/bin/bash


CURRENT_USER=$(whoami)


if [ "$CURRENT_USER" == "admin" ]; then
    echo "Access granted: Administrator privileges"
else
    echo "Access denied: Insufficient permissions"
    exit 1
fi
```

Combinare cu bucle pentru filtrare:

```
#!/bin/bash


for FILE in *.csv
do
    if echo $FILE | grep -q "sales"; then
        echo "Sales report detected: $FILE"
        mv $FILE /archive/sales/
    fi
done
```

7. Tehnici de depanare

7.1 Mod verbose (set -x)

Activarea modului de tracing afișează fiecare comandă cu substituțiile efectuate:

```
#!/bin/bash


set -x  # Activare debugging


for FILE in *.log

do

    echo $FILE

done
```

Output va include prefixul + pentru comenzile evaluate:

```
+ for FILE in *.log

+ echo access.log

access.log
```

7.2 Verificare context operațional

Inserarea de comenzi diagnostice clarifică contextul de execuție:

```
#!/bin/bash


echo "[DEBUG] Working directory: $(pwd)"

echo "[DEBUG] Current user: $(whoami)"

echo "[DEBUG] Hostname: $(hostname)"

echo "[DEBUG] Available files: $(ls)"
```

7.3 Analiză pipeline-uri complexe

Pentru comenzi cu multiple pipe-uri, testați incremental fiecare segment:

Comandă completă:

```
ps aux --sort=-%mem | head -15 | awk '{print $1, $4}' | column -t
```

Testare pas cu pas:

```
# Pas 1

ps aux --sort=-%mem



# Pas 2

ps aux --sort=-%mem | head -15



# Pas 3

ps aux --sort=-%mem | head -15 | awk '{print $1, $4}'
```

7.4 Ghilimele vs apostroafe

Reguli esențiale:

- `Ghilimele duble (")` — permit expandarea variabilelor
- `Apostroafe (')` — tratează totul ca text literal

Exemplu problematic:

```
#!/bin/bash



FILES="report.pdf data.csv"

ls "$FILES"  # Caută UN fișier numit 'report.pdf data.csv'
```

Soluție corectă:

```
ls $FILES    # Expandează ca două argumente separate
```

Blocarea evaluării cu apostroafe:

```
#!/bin/bash



VAR="test"

echo "$VAR"   # Output: test

echo '$VAR'   # Output: $VAR
```

8. Recomandări finale

1. **Validare date de intrare:** Verificați întotdeauna existența fișierelor și permisiunile necesare
2. **Logging comprehensiv:** Înregistrați operații critice cu timestamp-uri pentru audit
3. **Gestionare erori:** Implementați verificări de status și strategii de exit corespunzătoare
4. **Documentare:** Mențineți comentarii actualizate reflectând modificările logice
5. **Testare:** Rulați script-uri în medii de test înainte de deployment în producție

Notă finală: Automatizarea eficientă necesită înțelegere profundă a contextului operațional și anticipare a cazurilor limită. Investiți timp în depanare metodică și validare riguroasă pentru script-uri robuste și mentenabile.