Shell Scripting

Unitat 7 - Nel Banqué Torné

---

1. Introducció

Què és una shell?

Una **shell** és l'intèrpret de comandes d'UNIX que permet als usuaris interactuar amb el nucli del sistema i executar ordres. Existeixen diversos tipus d'intèrprets de comandes:

- **sh**: Bourne Shell (Stephen Bourne).
- **csh**: C Shell (Bill Joy).
- **ksh**: Korn Shell (David Korn).
- **bash**: Bourne Again Shell (Brian Fox i Chet Ramey).
- **zsh**: Z Shell (Paul Falstad).
- **tcsh**: Tenex C Shell (Ken Greer).

---

2. Comandes d'un Sistema Basat en UNIX

Una **comanda** és una seqüència de paraules separades per espais. Es compon de:

1. **Nom de la comanda**.
2. **Arguments**:
    - **Opcions**: Comencen amb - i modifiquen el comportament de la comanda.
    - **Paràmetres**: Informació addicional per executar la comanda.

Exemple:

```
cat /etc/shells  # Retorna la llista de shells disponibles al sistema.
```

Per consultar opcions disponibles: comanda --help

---

3. Execució de Comandes

Quan s'executa una comanda:

- Es crea un **nou procés**.
- Es retorna un **valor** que indica l'estat:
    - **0**: Èxit.
    - **Diferent de 0**: Error.

Exemple:

cat /etc/shells # Retorna 0 (execució correcta).
cat /etc/shell # Retorna diferent de 0 (fitxer no existeix).

---

4. Execució d'un Shell Script

Un **script de Bash** és un fitxer amb una seqüència de comandes.

Exemple d'Script:

Fitxer: list_shells.sh

#!/bin/bash
cat /etc/shells

Passos per Executar-lo:

1. **Amb Bash:** bash list_shells.sh
2. **Afegint permisos d'execució:** chmod +x list_shells.sh | ./list_shells.sh

---

5. Objectius del Shell Scripting

1. Automatitzar tasques repetitives.
2. Desenvolupar programes senzills i complexos.
3. Crear comandes personalitzades.
4. Generar scripts per a instal·lacions, configuracions i manteniments.

Exemple d'Instal·lació de gcc des del Codi Font:

```
VERSION=9.3.0     # Versió del compilador
wget https://ftp.gnu.org/gnu/gcc/gcc-$VERSION/gcc-$VERSION.tar.gz
tar -xzf gcc-$VERSION.tar.gz
mkdir gcc-$VERSION-build
cd gcc-$VERSION-build
../gcc-$VERSION/configure --enable-languages=c,c++ --disable-multilib
make -j$(nproc)
make install
```

---

6. Comanda echo

La comanda echo mostra text per pantalla.

Exemples:

```
echo "Hola, món!"
echo -n "Sense línia nova"
```

- **Cometes simples (')**: No permeten interpolació (expansió) de variables.
- **Cometes dobles (")**: Sí permeten interpolació (expansió) de variables. (Mantinc la forma original).
- **Sense cometes**: Es detecta com a coses independents

(Passa amb comandes que permeten Strings).

---

7. Fitxers d'Inicialització

Bash utilitza fitxers per establir l'entorn.

Globals:

- /etc/profile
- /etc/bashrc

Privats:

- ~: Carpeta d'usuari
- ~/.bash_profile: Execució al iniciar sessió.
- ~/.bashrc: Execució amb cada sessió interactiva.
- ~/.bash_logout: Execució al tancar sessió.

Exemple de Configuració:

```
echo "export EDITOR=nano" >> ~/.bashrc
echo "export PATH=$PATH:/home/usuari/bin" >> ~/.bashrc
```

---

8. Variables

Assignació de Variables:

```
nom="Bulbasaur"
echo $nom  # Mostra: Bulbasaur
# $ sempre indica que es una variable. Si no ho declarem, sempre serà String
# SEMPRE sense espais, bash no ho permet
```

També tenim alias que guarda comandes en noms posats per l'usuari.

Unset:

```
unset nom  # Buida la variable
# TAMBÉ puc posar un altre nom a nom, per canviar el valor de la variable
```

Enters (let, expr)

let: En una operació, interpreta els operands com a expressions aritmètiques i modifica el valor de les variables directament, sense imprimir el resultat. Específicament, funciona dins de scripts Bash i permet no usar el signe $ davant de les variables.

expr: En una operació, interpreta els operands com a expressions aritmètiques o manipulacions de cadenes. A DIFERÈNCIA de let, **expr** imprimeix el resultat directament a la sortida estàndard, en lloc de modificar variables internament. També cal utilitzar $ per referir-se a variables dins de l'expressió.

($# Retorna enter)

$(( )) Per a operacions

Amb $((...)) podem calcular directament lo que hi ha a dins dels parèntesis sense usar let.

Operacions amb arrays

- Per declarar arrays: pokedex=("Pikachu" "Charmander" "Bulbasaur")
- Per mostrar: echo ${pokedex[0]} # Mostra "Pikachu"
- Per actualitzar arrays ho fem igual que sempre
- Per recorrer array: for pokemon in "$(pokedex[0])"
- Podem definir arrays constants amb readonly
- $@ Retorna vector d'arguments

Operacions amb strings

- Per declarar strings: string=“Hello world!’’
- Reemplaçar: replace_first=${string/world/Earth}, hi ha més mètodes per reemplaçar. echo $replace_first # Mostra "Hello Earth!"
- Eliminar part del string: remove_part=${string/world/} echo $remove_part # Mostra "Hello !
- Substituir amb variables: old="world" new="Earth" replace_with_vars=${string/$old/$new} echo $replace_with_vars # Mostra "Hello Earth!"

9. Condicions if i else

Els condicionals permeten executar codi només si es compleix una condició.

Exemple de if

```
x=10
if [ $x -gt 5 ]  # Comprova si x és més gran que 5
then
  echo "x és més gran que 5"
fi
```

Exemple de if amb else

```
x=3
if [ $x -gt 5 ]
then
  echo "x és més gran que 5"
else
  echo "x no és més gran que 5"
fi
```

Exemple amb elif (condicions addicionals)

```
x=7
if [ $x -gt 10 ]
then
  echo "x és més gran que 10"
elif [ $x -gt 5 ]
then
  echo "x és més gran que 5 però no més gran que 10"
else
  echo "x no és més gran que 5"
fi
```

---

10. Bucles for, while i until

Els bucles permeten repetir una acció diverses vegades.

Bucle for

Iterar sobre una llista de valors

```
for pokemon in "Pikachu" "Charmander" "Bulbasaur"
do
  echo $pokemon
done
```

Iterar sobre un rang de números

```
for i in {1..5}
do
  echo $i
done
```

Bucle while

Repeteix l'acció mentre una condició sigui certa.

```
counter=1
while [ $counter -le 5 ]  # Comprova si el counter és menor o igual a 5
do
  echo $counter
  ((counter++))  # Incrementa el counter
done
```

Bucle until

Repeteix l'acció fins que una condició sigui certa.

```
counter=1
until [ $counter -gt 5 ]  # Repeteix fins que el counter sigui més gran que 5
do
  echo $counter
  ((counter++))  # Incrementa el counter
done
```

---

11. Funcions

Les funcions permeten agrupar codi per reutilitzar-lo en diferents parts d'un script.

Declarar i cridar una funció

```
# Declaració d'una funció
saludar() {
  echo "Hola, $1!"  # Imprimeix un missatge amb el primer argument
}

# Crida a la funció
saludar "Món"  # Mostra "Hola, Món!"
```

Funció amb múltiples arguments

```
multiplicar() {
  resultat=$(( $1 * $2 ))  # Multiplica els dos arguments
  echo "El resultat de multiplicar $1 per $2 és $resultat"
}

multiplicar 3 5  # Mostra "El resultat de multiplicar 3 per 5 és 15"
```

Resum

- **if / else**: Permet executar codi basat en condicions.
- **for**: Itera sobre una llista o un rang de valors.
- **while**: Repeteix mentre una condició sigui certa.
- **until**: Repeteix fins que una condició sigui certa.
- **Funcions**: Agrupen codi reutilitzable i poden acceptar arguments.