
Shell Scripts

Índex

1. Introducció	1
2. Shells disponibles	1
3. Estructura bàsica d'un script	1
3.1 El shebang	1
3.2 Estructura típica	1
3.3 Permisos d'execució	2
4. Variables	2
4.1 Declaració i assignació	2
4.2 Ús de variables	3
4.3 Variables especials	3
4.4 Substitució de variables	3
4.5 Variables d'entorn habituals	4
5. Entrada i sortida	4
5.1 echo i printf	4
5.2 Lectura d'entrada: read	5
5.3 Redirecció	5
6. Operadors i expressions	6
6.1 Operadors aritmètics	6
6.2 Operadors de comparació numèrica	6
6.3 Operadors de comparació de cadenes	6
6.4 Operadors de fitxers	7
6.5 Operadors lògics	7
7. Estructures de control	8
7.1 if / elif / else	8
7.2 case	8
7.3 for	9
7.4 while	10
7.5 until	10
7.6 break i continue	10
8. Funcions	11
Exemple complet amb funcions	11
9. Arrays	12
10. Substitució d'ordres	12
11. Gestió d'errors i robustesa	13
11.1 Opcions de seguretat	13
11.2 Trampes (trap)	13
11.3 Codis de retorn	13
11.4 Fitxers de bloqueig (lock)	14
12. Processament de text	14
13. Processament d'arguments	15
13.1 Manual amb \$1, \$2...	15
13.2 Amb getopt s	15
14. Exemple complet: script de còpia de seguretat	16
15. Bones pràctiques	17
16. Depuració	18
17. Resum de la sintaxi	18
18. Referències	19

1. Introducció

Un **shell script** és un fitxer de text que conté una seqüència d'ordres del shell (intèrpret d'ordres) que s'executen de manera seqüencial. Permeten automatitzar tasques repetitives, combinar ordres complexes i crear eines personalitzades per a l'administració del sistema.

Els shell scripts són la base de l'automatització en sistemes Unix/Linux i constitueixen una habilitat fonamental per a qualsevol administrador de sistemes.

2. Shells disponibles

Shell	Ruta	Descripció
sh	/bin/sh	Bourne Shell, el més bàsic i portable
bash	/bin/bash	Bourne Again Shell, el més estès a Linux
dash	/bin/dash	Shell lleugera, compatible POSIX, usada al boot
zsh	/bin/zsh	Shell avançada, molt configurable
fish	/usr/bin/fish	Shell amigable i interactiva
ksh	/bin/ksh	Korn Shell, popular en entorns UNIX

La majoria de scripts s'escriuen per a **bash**, que és la shell per defecte a la majoria de distribucions Linux.

3. Estructura bàsica d'un script

3.1 El shebang

La primera línia d'un script especifica quin intèrpret l'ha d'executar:

```
#!/bin/bash
```

Altres exemples:

```
#!/bin/sh          # Shell POSIX genèrica
#!/usr/bin/env bash # Cerca bash al PATH (més portable)
#!/usr/bin/python3 # Script Python
```

3.2 Estructura típica

```
#!/bin/bash
# =====
# Nom:      script_exemple.sh
# Descripció: Exemple d'estructura d'un script bash
# Autor:    Ramon
# Data:     2025-06-09
# Versió:   1.0
# =====
```

```
# --- Variables globals ---
NOM_SCRIPT=$(basename "$0")
VERSIÓ="1.0"
LOG="/var/log/script.log"

# --- Funcions ---
mostrar_ajuda() {
    echo "Ús: $NOM_SCRIPT [opcions]"
    echo "  -h  Mostra aquesta ajuda"
    echo "  -v  Mostra la versió"
}

# --- Codi principal ---
echo "Iniciant $NOM_SCRIPT versió $VERSIÓ"
```

3.3 Permisos d'execució

```
# Donar permís d'execució
chmod +x script.sh
chmod 755 script.sh

# Executar
./script.sh
bash script.sh
sh script.sh
```

4. Variables

4.1 Declaració i assignació

```
# Assignació (sense espais al voltant de =)
nom="Ramon"
edat=42
ruta="/home/ramon"

# Majúscules per a constants (convenció)
MAX_INTENTS=3
COLOR_VERD="\033[0;32m"
```

4.2 Ús de variables

```
echo "$nom"
echo "${nom}"          # Forma preferida (evita ambigüitats)
echo "Hola, ${nom}!"

# Concatenació
prefix="fitxer"
sufix=".txt"
nom_fitxer="${prefix}_001${sufix}"
```

4.3 Variables especials

Variable	Descripció
\$0	Nom del script
\$1--\$9	Arguments posicionals (de l'1 al 9)
\${10}	Arguments a partir del 10 (calen claus)
\$#	Nombre d'arguments passats
\$@	Tots els arguments com a llista separada
\$*	Tots els arguments com una sola cadena
?	Codi de retorn de l'última ordre executada
\$\$	PID del procés actual (del script)
\$!	PID de l'últim procés en segon pla
_	Últim argument de l'ordre anterior
\$-	Opcions actives del shell

```
#!/bin/bash
echo "Script: $0"
echo "Primer argument: $1"
echo "Nombre d'arguments: $#"
```

```
echo "Tots els arguments: $@"
```

4.4 Substitució de variables

```
# Valor per defecte si la variable és buida o no definida
echo "${nom:-"Desconegut"}"

# Assignar valor per defecte si és buida
echo "${nom:="Anònim"}"

# Error si la variable és buida
echo "${nom:?La variable nom és obligatòria}"

# Valor alternatiu si la variable té valor
echo "${nom:+La variable nom té valor}"

# Longitud de la variable
echo "${#nom}"
```

```
# Subcadena: ${variable:inici:longitud}
cadena="Hola món"
echo "${cadena:5:3}"      # "món"

# Eliminar prefix (patró més curt)
fitxer="document.txt"
echo "${fitxer#*.}"       # "txt"

# Eliminar sufix (patró més curt)
echo "${fitxer%.*}"       # "document"
```

4.5 Variables d'entorn habituals

```
$HOME      # Directori personal de l'usuari
$USER      # Nom de l'usuari actual
$PATH      # Rutes de cerca d'executables
$PWD       # Directori de treball actual
$OLDPWD    # Directori anterior
$SHELL     # Shell actual
$HOSTNAME  # Nom de la màquina
$LANG      # Configuració regional
$EDITOR    # Editor de text per defecte
$TERM      # Tipus de terminal
```

5. Entrada i sortida

5.1 echo i printf

```
# echo bàsic
echo "Hola món"
echo -n "Sense salt de línia"
echo -e "Amb\ttabulació\ni salt de línia"

# printf (més precís)
printf "Nom: %s, Edat: %d\n" "$nom" "$edat"
printf "%-20s %5d\n" "Article" 42      # Alineació
```

5.2 Lectura d'entrada: read

```
# Llegir una línia
read -p "Introdueix el teu nom: " nom
echo "Hola, $nom!"

# Llegir sense mostrar (contrasenyes)
read -sp "Contrasenya: " contrasenya
echo ""

# Llegir amb temps límit
read -t 10 -p "Resposta (10s): " resposta

# Llegir en un array
read -a colors <<< "vermell verd blau"
echo "${colors[0]}" # vermell

# Llegir línia a línia d'un fitxer
while IFS= read -r linia; do
    echo "$linia"
done < fitxer.txt
```

5.3 Redirecció

```
# Sortida estàndard a fitxer
echo "text" > fitxer.txt      # Sobreescriu
echo "text" >> fitxer.txt     # Afegeix

# Entrada des de fitxer
sort < llista.txt

# Errors a fitxer
ordre 2> errors.log
ordre > sortida.log 2>&1      # Tot (stdout i stderr) al mateix fitxer
ordre > sortida.log 2> errors.log # Separats

# Descartar sortida
ordre > /dev/null 2>&1

# Here document
cat << EOF
Línia 1
Línia 2
$variable_expandida
EOF

# Here string
grep "paraula" <<< "aquesta és la cadena on buscar"
```

6. Operadors i expressions

6.1 Operadors aritmètics

```
# Amb (( ))
((resultat = 5 + 3))
echo $((10 * 3 - 2))
echo $((2 ** 8))          # Potència: 256
echo $((17 % 5))          # Mòdul: 2

# Amb let
let "x = 5 + 3"

# Amb expr (més antic)
resultat=$(expr 5 + 3)

# Amb bc (per a decimals)
resultat=$(echo "scale=2; 10/3" | bc)
```

6.2 Operadors de comparació numèrica

Operador	Significat	Exemple
-eq	Igual	[\$a -eq \$b]
-ne	Diferent	[\$a -ne \$b]
-lt	Menor que	[\$a -lt \$b]
-le	Menor o igual	[\$a -le \$b]
-gt	Major que	[\$a -gt \$b]
-ge	Major o igual	[\$a -ge \$b]

6.3 Operadors de comparació de cadenes

Operador	Significat	Exemple
=	Igual	["\$a" = "\$b"]
!=	Diferent	["\$a" != "\$b"]
<	Menor (ordre lexicogràfic)	[["\$a" < "\$b"]]
>	Major (ordre lexicogràfic)	[["\$a" > "\$b"]]
-z	Cadena buida	[-z "\$a"]
-n	Cadena no buida	[-n "\$a"]
=~	Coincidència amb regex	[["\$a" =~ ^[0-9]+\$]]

6.4 Operadors de fitxers

Operador	Condicció verdadera si...
-e	El fitxer existeix
-f	Existeix i és un fitxer regular
-d	Existeix i és un directori
-l	Existeix i és un enllaç simbòlic
-r	Existeix i és llegible
-w	Existeix i és escriuible
-x	Existeix i és executable
-s	Existeix i no és buit (mida > 0)
-b	Existeix i és un dispositiu de blocs
-c	Existeix i és un dispositiu de caràcters
-p	Existeix i és una canonada (pipe)
f1 -nt f2	f1 és més nou que f2
f1 -ot f2	f1 és més vell que f2

6.5 Operadors lògics

```
# AND
[ $a -gt 0 ] && [ $b -gt 0 ]
[[ $a -gt 0 && $b -gt 0 ]]

# OR
[ $a -gt 0 ] || [ $b -gt 0 ]
[[ $a -gt 0 || $b -gt 0 ]]

# NOT
[ ! -f fitxer.txt ]
```

7. Estructures de control

7.1 if / elif / else

```
if [ condició ]; then
    # ordres
elif [ altra_condició ]; then
    # ordres
else
    # ordres
fi

# Exemples
if [ -f "/etc/passwd" ]; then
    echo "El fitxer existeix"
fi

if [[ "$usuari" == "root" ]]; then
    echo "Ets root"
elif [[ "$usuari" == "ramon" ]]; then
    echo "Hola Ramon"
else
    echo "Usuari desconegut"
fi

# Forma compacta amb && i ||
[ -d "/tmp/dades" ] || mkdir "/tmp/dades"
[ "$edat" -ge 18 ] && echo "Major d'edat" || echo "Menor d'edat"
```

7.2 case

```
case "$variable" in
    patró1)
        # ordres
        ;;
    patró2|patró3)
        # ordres per a dos patrons
        ;;
    patró*)
        # comodí
        ;;
    *)
        # per defecte
        ;;
esac

# Exemple
case "$1" in
    start)
        echo "Iniciant servei..."
        ;;
```

```

    stop)
        echo "Aturant servei..."
        ;;
    restart|reload)
        echo "Reiniciant servei..."
        ;;
    *)
        echo "Ús: $0 {start|stop|restart}"
        exit 1
        ;;
esac

```

7.3 for

```

# Iterar sobre una llista
for element in un dos tres quatre; do
    echo "$element"
done

# Iterar sobre fitxers
for fitxer in /etc/*.conf; do
    echo "Fitxer: $fitxer"
done

# Bucle numèric estil C
for ((i=0; i<10; i++)); do
    echo "Iteració $i"
done

# Amb seq
for i in $(seq 1 10); do
    echo "$i"
done

# Iterar sobre arguments
for arg in "$@"; do
    echo "Argument: $arg"
done

# Iterar sobre línies d'un fitxer
for linia in $(cat fitxer.txt); do
    echo "$linia"
done

```

7.4 while

```
# Bucle while bàsic
comptador=0
while [ $comptador -lt 5 ]; do
    echo "Comptador: $comptador"
    ((comptador++))
done

# Llegir fitxer línia a línia (forma correcta)
while IFS= read -r linia; do
    echo "$linia"
done < fitxer.txt

# Bucle infinit amb sortida
while true; do
    read -p "Ordre (q per sortir): " ordre
    [ "$ordre" = "q" ] && break
    echo "Has escrit: $ordre"
done

# Llegir fins a EOF
while read -r linia; do
    processa "$linia"
done
```

7.5 until

```
# Executa mentre la condició sigui FALSA
fins=10
until [ $fins -le 0 ]; do
    echo "Compte enrere: $fins"
    ((fins--))
done
```

7.6 break i continue

```
for i in $(seq 1 10); do
    [ $i -eq 5 ] && continue    # Salta la iteració 5
    [ $i -eq 8 ] && break      # Surt del bucle a la 8
    echo "$i"
done
```

8. Funcions

```
# Declaració
nom_funcio() {
    # cos de la funció
    echo "Primer argument: $1"
    return 0    # Codi de retorn (0--255)
}

# Declaració alternativa
function nom_funcio {
    echo "Funció"
}

# Crida
nom_funcio "argument1" "argument2"

# Capturar el valor de retorn
nom_funcio
resultat=$?

# Capturar la sortida de la funció
sortida=$(nom_funcio "arg")
```

Exemple complet amb funcions

```
#!/bin/bash

# Colors
VERMELL='\033[0;31m'
VERD='\033[0;32m'
GROC='\033[1;33m'
NC='\033[0m'    # No Color

log_info() { echo -e "${VERD}[INFO]${NC} $1"; }
log_warn() { echo -e "${GROC}[WARN]${NC} $1"; }
log_error() { echo -e "${VERMELL}[ERROR]${NC} $1" >&2; }

comprovar_root() {
    if [[ $EUID -ne 0 ]]; then
        log_error "Aquest script requereix privilegis de root"
        exit 1
    fi
}

comprovar_dependencia() {
    local prog="$1"
    if ! command -v "$prog" &>/dev/null; then
        log_error "No s'ha trobat '$prog'. Instal·la'l primer."
        return 1
    fi
}
```

```

    return 0
}

comprovar_root
comprovar_dependencia rsync || exit 1
log_info "Totes les dependències trobades"

```

9. Arrays

```

# Declaració
colors=("vermell" "verd" "blau")
declare -a nombres=(1 2 3 4 5)

# Accés
echo "${colors[0]}"          # vermell
echo "${colors[@]}"          # tots els elements
echo "${#colors[@]}"         # nombre d'elements
echo "${!colors[@]}"         # índexs

# Modificació
colors[3]="groc"
colors+=("taronja")          # Afegir element

# Iterar
for color in "${colors[@]}"; do
    echo "$color"
done

# Arrays associatius (bash 4+)
declare -A capitals
capitals["Catalunya"]="Barcelona"
capitals["França"]="París"
echo "${capitals["Catalunya"]}"
for pais in "${!capitals[@]}"; do
    echo "$pais → ${capitals[$pais]}"
done

```

10. Substitució d'ordres

```

# Forma moderna (preferida)
data=$(date +%Y-%m-%d)
usuari=$(cat /etc/passwd | cut -d: -f1)
num_fitxers=$(ls /etc | wc -l)

# Forma antiga (backticks)
data=`date +%Y-%m-%d`

# Anidada

```

```
resultat=$(echo $(date +%Y) + 1 | bc)
```

11. Gestió d'errors i robustesa

11.1 Opcions de seguretat

```
#!/bin/bash
set -e          # Surt si qualsevol ordre falla (codi ≠ 0)
set -u          # Surt si s'usa una variable no definida
set -o pipefail # Falla si qualsevol part d'un pipe falla
set -x          # Mode debug: mostra cada ordre abans d'executar-la

# Forma compacta habitual
set -euo pipefail
```

11.2 Trampes (trap)

```
# Executar una funció en sortir (neteja)
netejar() {
    echo "Netejant fitxers temporals..."
    rm -f /tmp/script_temp_$$
}
trap netejar EXIT

# Capturar Ctrl+C
trap "echo 'Interromput per l usuari'; exit 1" INT

# Capturar múltiples senyals
trap netejar EXIT INT TERM

# Ignorar una senyal
trap "" HUP
```

11.3 Codis de retorn

```
# Conveni: 0 = èxit, 1-255 = error
commanda_que_pot_fallar
if [ $? -ne 0 ]; then
    echo "Error en l'execució"
    exit 1
fi

# Forma compacta
commanda_que_pot_fallar || { echo "Error"; exit 1; }

# Funció amb codi de retorn
```

```

dividir() {
    if [ "$2" -eq 0 ]; then
        echo "Error: divisió per zero" >&2
        return 1
    fi
    echo $(( $1 / $2 ))
    return 0
}

```

11.4 Fitxers de bloqueig (lock)

```

LOCK="/tmp/script.lock"

if [ -e "$LOCK" ]; then
    echo "El script ja s'està executant (PID: $(cat $LOCK))"
    exit 1
fi

echo $$ > "$LOCK"
trap "rm -f $LOCK" EXIT

```

12. Processament de text

```

# grep: cercar patrons
grep "error" /var/log/syslog
grep -i "error" fitxer.txt          # Sense distinció
    ↪ majúscules/minúscules
grep -r "funció" /home/ramon/      # Recursiu
grep -v "comentari" fitxer.txt     # Línies que NO coincideixen
grep -c "error" fitxer.txt         # Comptar coincidències
grep -n "error" fitxer.txt         # Mostrar número de línia

# sed: editar flux de text
sed 's/vell/nou/g' fitxer.txt      # Substituir
sed -i 's/vell/nou/g' fitxer.txt   # Substituir en el fitxer
sed -n '5,10p' fitxer.txt          # Mostrar línies 5-10
sed '/patró/d' fitxer.txt           # Eliminar línies amb patró

# awk: processament de camps
awk '{print $1}' fitxer.txt         # Primer camp
awk -F: '{print $1, $3}' /etc/passwd # Camps 1 i 3 amb separador ':'
awk '$3 > 1000 {print $1}' /etc/passwd # Filtre + acció
awk 'BEGIN{suma=0} {suma+=$1} END{print suma}' nombres.txt

# cut: extreure camps
cut -d: -f1,3 /etc/passwd
cut -c1-10 fitxer.txt              # Caràcters 1-10

```

```
# sort i uniq
sort fitxer.txt
sort -n fitxer.txt          # Ordenació numèrica
sort -r fitxer.txt          # Ordre invers
sort -u fitxer.txt          # Eliminar duplicats
sort fitxer.txt | uniq -c    # Comptar ocurrences

# tr: traduir o eliminar caràcters
echo "Hola Món" | tr '[:upper:]' '[:lower:]'
echo "a:b:c" | tr ':' '\n'
echo "text" | tr -d 'aeiou'  # Eliminar vocals
```

13. Processament d'arguments

13.1 Manual amb \$1, \$2...

```
#!/bin/bash
if [ $# -lt 2 ]; then
    echo "Ús: $0 origen destí"
    exit 1
fi
origen="$1"
desti="$2"
```

13.2 Amb getopt

```
#!/bin/bash
mostrar_ajuda() {
    echo "Ús: $0 [-h] [-v] [-f fitxer] [-n nom]"
}

verbos=false
fitxer=""
nom=""

while getopt "hvf:n:" opcio; do
    case "$opcio" in
        h) mostrar_ajuda; exit 0 ;;
        v) verbos=true ;;
        f) fitxer="$OPTARG" ;;
        n) nom="$OPTARG" ;;
        ?) mostrar_ajuda; exit 1 ;;
    esac
done

# Desplaçar arguments processats
shift $((OPTIND - 1))
```

```
$verbos && echo "Mode verbós activat"
echo "Fitxer: $fitxer"
echo "Nom: $nom"
echo "Arguments restants: $@"
```

14. Exemple complet: script de còpia de seguretat

```
#!/bin/bash
# =====
# backup.sh - Script de còpia de seguretat
# Ús: ./backup.sh [-v] [-d directori_destí] directori_origen
# =====

set -euo pipefail

# --- Constants ---
DATA=$(date +%Y-%m-%d_%H-%M-%S)
LOG_DIR="/var/log/backups"
LOG_FILE="${LOG_DIR}/backup_${DATA}.log"

# --- Colors ---
VERD='\033[0;32m'; VERMELL='\033[0;31m'; NC='\033[0m'

# --- Funcions ---
log() { echo -e "[$(date +%H:%M:%S)] $1" | tee -a "$LOG_FILE"; }
ok() { log "${VERD}[OK]${NC} $1"; }
err() { log "${VERMELL}[ERROR]${NC} $1" >&2; }

netejar() { log "Script finalitzat."; }
trap netejar EXIT

comprovar_eines() {
    for eina in rsync df; do
        command -v "$eina" &>/dev/null || { err "Falta '$eina'"; exit
↵ 1; }
    done
}

# --- Processament d'arguments ---
verbos=false
desti="/backup"

while getopts "vd:h" opt; do
    case "$opt" in
        v) verbos=true ;;
        d) desti="$OPTARG" ;;
        h) echo "Ús: $0 [-v] [-d destí] origen"; exit 0 ;;
        *) exit 1 ;;
    esac
done
```

```

shift $((OPTIND - 1))

[ $# -eq 0 ] && { err "Cal especificar el directori origen"; exit 1; }
origen="$1"

# --- Validació ---
[ -d "$origen" ] || { err "L'origen '$origen' no existeix"; exit 1; }
[ -d "$desti" ] || mkdir -p "$desti"
[ -d "$LOG_DIR" ] || mkdir -p "$LOG_DIR"

comprovar_eines

# --- Còpia ---
opcions_rsync="-a --delete"
$verbos && opcions_rsync="$opcions_rsync -v"

log "Iniciant backup: $origen → $desti"
# shellcheck disable=SC2086
rsync $opcions_rsync "$origen/" "${desti}/" 2>> "$LOG_FILE"
ok "Backup completat correctament"
log "Log desat a: $LOG_FILE"

```

15. Bones pràctiques

1. **Shebang sempre:** indicar explícitament l'interpret (`#!/bin/bash`).
2. **set -euo pipefail:** activar les opcions de seguretat al principi.
3. **Cometes dobles:** usar sempre `"$variable"` per evitar problemes amb espais.
4. **Rutes absolutes:** en scripts automatitzats, no assumir el PATH.
5. **Codis de retorn:** verificar sempre `$?` o usar `||` per gestionar errors.
6. **trap:** netejar recursos temporals en sortir.
7. **Comentaris:** documentar el propòsit, els arguments i les decisions importants.
8. **shellcheck:** usar l'eina d'anàlisi estàtica per detectar errors:

```
shellcheck script.sh
```

9. **Noms descriptius:** variables i funcions amb noms clars.
10. **Funcions per a codi reutilitzable:** no repetir blocs de codi.
11. **Sortida d'errors a stderr:** `echo "Error" >&2`.
12. **Fitxers temporals segurs:** `bash tmp=$(mktemp) trap "rm -f $tmp" EXIT`

16. Depuració

```
# Mode debug: mostra cada ordre
bash -x script.sh
set -x    # Activar dins l'script
set +x    # Desactivar

# Mode verbose: mostra cada línia
bash -v script.sh

# Comprovar sintaxi sense executar
bash -n script.sh

# shellcheck: anàlisi estàtica
shellcheck script.sh
```

17. Resum de la sintaxi

```
# Variables
var="valor"
echo "$var"

# Condicional
if [ condició ]; then ...; fi

# Bucle for
for i in llista; do ...; done

# Bucle while
while [ condició ]; do ...; done

# Funció
funcio() { ...; }

# Substitució d'ordres
var=${ordre}

# Aritmètica
((x = a + b))
echo ${a * b}

# Array
arr=(a b c)
echo "${arr[0]}"

# Case
case "$var" in patró) ... ;; esac
```

18. Referències

- `man bash` --- Manual complet de bash
- `man test` --- Operadors de condicions
- [Bash Guide for Beginners \(TLDP\)](#)
- [Advanced Bash-Scripting Guide \(TLDP\)](#)
- shellcheck.net --- Anàlisi estàtica en línia
- [Arch Wiki: Bash](#)

Versions d'aquest document

- [HTML](#)
- [PDF](#)
- [ODT](#)
- [MD](#)

[Domini Públic \(CC0\)](#)