Bash Scripting_

Guía de atajos · automatiza tareas en Linux

```
user@dajomar:~$ ./script.sh
```

Tu primer script

```
#!/usr/bin/env bash
set -euo pipefail  # modo estricto

echo "Hola, $USER. Hoy es $(date +%F)"
```

```
chmod +x script.sh
./script.sh
bash -x script.sh  # depurar línea a línea
```

`#!`	Shebang: indica qué intérprete usar
`set -e`	Se detiene si un comando falla
`set -u`	Error si usas una variable sin definir
`set -o pipefail`	Falla si falla cualquier parte de un pipe

Bash no es lo mismo que sh:

`sh` es el estándar POSIX básico y Bash lo amplía. En Ubuntu, `/bin/sh` es **dash** y en Alpine es **ash**, así que `[[ ]]`, los arrays o `{1..5}` fallan ahí.

```
#!/usr/bin/env bash  # usas funciones de Bash
#!/bin/sh            # solo POSIX, corre en todos
ls -l /bin/sh        # ¿qué shell es tu sh?
```

Variables

```
nombre="David"  # sin espacios junto al =
echo "Hola $nombre"
echo "Archivo: ${nombre}_backup.tar"
hoy=$(date +%Y-%m-%d)  # resultado de un comando
total=$(( 5 + 3 * 2 ))          # aritmética
readonly RUTA="/opt/app"        # constante
export ENTORNO="prod"  # visible para subprocesos
read -p "¿Servidor? " servidor  # pedir dato
```

Usa siempre comillas dobles: `"$var"`. Sin ellas, un valor con espacios se rompe en varias palabras.

Variables especiales

`$0`	Nombre del script
`$1, $2 …`	Argumentos recibidos
`$#`	Cantidad de argumentos
`"$@"`	Todos los argumentos, uno por uno
`$?`	Código de salida del último comando (0 = OK)
`$$`	PID del script
`$!`	PID del último proceso en segundo plano

```
if [[ $# -lt 1 ]]; then
  echo "Uso: $0 <directorio>" >&2
  exit 1
fi
```

Condicionales

```
if [[ -f "$archivo" ]]; then
  echo "Existe"
elif [[ -d "$archivo" ]]; then
  echo "Es un directorio"
else
  echo "No existe"
fi
```

Prueba	Verdadero si…
`-f / -d`	Es archivo / es directorio
`-e`	Existe (cualquier tipo)
`-r -w -x`	Se puede leer / escribir / ejecutar
`-z / -n`	Texto vacío / no vacío
`== / !=`	Textos iguales / distintos
`=~`	Coincide con una regex
`-eq -ne`	Números iguales / distintos
`-lt -le`	Menor / menor o igual
`-gt -ge`	Mayor / mayor o igual
`&& || !`	Y, O, NO

Prefiere `[[ ]]` sobre `[ ]`: maneja mejor espacios y permite `&&`, `||` y regex.

case: varias opciones

```
case "$1" in
  start)   systemctl start app ;;
  stop)    systemctl stop app ;;
  restart) systemctl restart app ;;
  *)       echo "Uso: $0 {start|stop|restart}"
           exit 1 ;;
esac
```

Bucles

```
for srv in web01 web02 db01; do
  ping -c1 -W1 "$srv" >/dev/null && echo "$srv OK"
done

for i in {1..5}; do echo "Intento $i"; done

for f in /var/log/*.log; do
  echo "$f: $(wc -l < "$f") líneas"
done
```

Leer un archivo línea por línea

```
while IFS= read -r linea; do
  echo "→ $linea"
done < servidores.txt
```

`break` sale del bucle · `continue` salta a la siguiente vuelta.

Funciones

```
log() {
  echo "[$(date '+%F %T')] $*" \
    | tee -a /var/log/mi.log
}

existe_usuario() {
  local usuario="$1"  # variable local
  id "$usuario" &>/dev/null
}

if existe_usuario "ana"; then log "ana existe"; fi
```

`return` devuelve un código (0-255), no texto. Para devolver texto usa `echo` y captúralo con `$(función)`.

Arrays

```
servidores=("web01" "web02" "db01")
echo "${servidores[0]}"   # primero
echo "${servidores[@]}"   # todos
echo "${#servidores[@]}"  # cantidad
servidores+=("cache01")   # agregar
for s in "${servidores[@]}"; do echo "$s"; done
```

Manipular texto

Expresión	Con archivo="reporte.tar.gz"
`${#archivo}`	Largo: 14
`${archivo%.gz}`	Quita al final: reporte.tar
`${archivo%%.*}`	Quita todo desde el 1er punto: reporte
`${archivo#*.}`	Quita al inicio: tar.gz
`${archivo/tar/zip}`	Reemplaza: reporte.zip.gz
`${archivo^^}`	Mayúsculas: REPORTE.TAR.GZ
`${var:-def}`	Usa "def" si var está vacía
`${var:?msg}`	Error con msg si var está vacía

Redirecciones y here-doc

```
cmd > salida.txt 2>&1  # salida y errores
cmd &>/dev/null        # silenciar todo
echo "error" >&2       # escribir en stderr

cat > /etc/motd <<EOF
Servidor: $(hostname)
Administrado por dajomar
EOF
```

Errores y limpieza

```
tmp=$(mktemp -d)
trap 'rm -rf "$tmp"' EXIT  # limpia al salir

if ! cp origen.txt "$tmp"/; then
  echo "Falló la copia" >&2
  exit 1
fi

comando || { echo "falló"; exit 1; }
```

Convención: `exit 0` todo bien · `exit 1` o más, error.

Programar con cron

```
crontab -e  # editar tareas
crontab -l  # listar
```

```
┌─ minuto (0-59)
│ ┌─ hora (0-23)
│ │ ┌─ día del mes (1-31)
│ │ │ ┌─ mes (1-12)
│ │ │ │ ┌─ día de la semana (0-7, 0 y 7 = dom)
* * * * *  comando
```

`0 2 * * *`	Todos los días a las 2:00
`*/15 * * * *`	Cada 15 minutos
`0 9 * * 1-5`	Lunes a viernes a las 9:00
`0 0 1 * *`	El día 1 de cada mes

En cron usa rutas absolutas y redirige la salida a un log: `… >> /var/log/backup.log 2>&1`

Ejemplo real: backup con rotación

```
#!/usr/bin/env bash
set -euo pipefail

ORIGEN="/var/www"
DESTINO="/backups"
DIAS=7
archivo="$DESTINO/www_$(date +%F_%H%M).tar.gz"

log() { echo "[$(date '+%F %T')] $*"; }

mkdir -p "$DESTINO"
log "Iniciando backup de $ORIGEN"
tar -czf "$archivo" "$ORIGEN"
tam=$(du -h "$archivo" | cut -f1)
log "Creado: $archivo ($tam)"

find "$DESTINO" -name 'www_*.tar.gz' \
  -mtime +"$DIAS" -delete
log "Backups con más de $DIAS días eliminados"
```

Pro Tip:
Pasa tus scripts por `shellcheck script.sh` antes de usarlos. Detecta comillas faltantes, variables sin usar y errores comunes que `bash` no te avisa.