



marlon - zsh - 80x24

```
~ > sw_vers -productName
macOS
~ > mdfind -onlyin ~ 'kMDItemFSSize > 500000000' | wc -l
12
~ > defaults write com.apple.finder AppleShowAllFiles -bool true
~ > caffeinate -dims & pbpaste | sort -u | pbcopy
~ > █
```

\$ man mac

La Consola del Mac

Comandos avanzados de la Terminal de macOS: zsh, automatización, sistema, redes y seguridad

Marlon Falcón Hernández

Implemento BIM 2.0 · Valencia

EDICIÓN 2026

La Consola del Mac. Comandos avanzados de la Terminal de macOS

© 2026 Marlon Falcón Hernández. Primera edición, octubre de 2026. Valencia, España.

Todos los comandos se han escrito para macOS reciente (Sonoma, Sequoia y posteriores) con zsh como shell por defecto. Algunas opciones cambian entre versiones del sistema: ante la duda, consulta `man comando` en tu propio equipo.

Los comandos que modifican el sistema (marcados con `sudo`) pueden alterar la configuración de tu Mac. Haz copia de seguridad con Time Machine antes de experimentar. El autor no se responsabiliza de daños derivados del uso de este libro.

macOS, Mac, Finder, Spotlight, Time Machine y Terminal son marcas de Apple Inc.

Contenido

- 00** **Antes de empezar**
Terminal, iTerm2, la anatomía de un comando y cómo leer el manual

- 01** **zsh a fondo**
Atajos, historial, expansiones, globbing extendido y modificadores

- 02** **Redirecciones y tuberías**
Descriptores, here-docs, sustitución de procesos, xargs y tee

- 03** **Buscar archivos y contenido**
find, mdfind, mdls, grep, ripgrep y fd

- 04** **Procesar texto como un profesional**
sed, awk, cut, sort, uniq, column, jq y plutil

- 05** **Los comandos exclusivos de macOS**
open, pbcopy, say, caffeinate, sips, screenshot, textutil y más

- 06** **defaults: los ajustes ocultos**
Leer, escribir y respaldar preferencias del sistema

- 07** **Procesos, memoria y rendimiento**
ps, top, lsof, fs_usage, vm_stat, powermetrics y log

- 08** **Discos, archivos y copias**
diskutil, hdiutil, ditto, rsync, tmutil y xattr

- 09** **Redes desde la consola**
networksetup, scutil, curl, networkQuality, dig, nc y ssh

- 10** **Seguridad y llavero**
security, codesign, spctl, csrutil, fdesetup y Touch ID para sudo

- 11** **Automatización**
Scripts zsh, osascript, Atajos, launchd y cron

- 12** **Tu entorno ideal**
Homebrew, .zshrc, alias, funciones y herramientas modernas

- A** **Chuleta final**
Los comandos de este libro en dos páginas

Antes de empezar

La Terminal no es una herramienta del pasado: es la interfaz más rápida y precisa que tiene tu Mac.

macOS es un sistema Unix certificado. Debajo del Finder, el Dock y las ventanas translúcidas hay un núcleo (XNU), un sistema de archivos (APFS) y cientos de utilidades de línea de comandos. Algunas son las clásicas de cualquier Unix (`ls`, `grep`, `find`) y otras son exclusivas de Apple (`defaults`, `mdfind`, `diskutil`, `pbcopy`). Este libro se centra en ambas, pero con una idea clara: que salgas de cada capítulo sabiendo hacer cosas que con el ratón serían lentas o directamente imposibles.

Abrir la Terminal

La forma más rápida es `⌘` + `Espacio`, escribir *Terminal* y pulsar `↵`. La app vive en `/Applications/Utilities/Terminal.app`. Muchos usuarios avanzados prefieren alternativas como iTerm2, Warp, Ghostty o WezTerm; todos los comandos de este libro funcionan igual en cualquiera de ellas, porque lo que ejecuta los comandos no es la ventana, sino la **shell**.

La shell: zsh

Desde macOS Catalina (2019), la shell por defecto es **zsh**. Si vienes de Linux, notarás que se parece mucho a `bash`, pero con mejor autocompletado, globbing más potente y modificadores de variables muy útiles. Comprueba la tuya:

```
> echo $SHELL
/bin/zsh
> zsh --version
zsh 5.9 (arm64-apple-darwin24.0)
```

Anatomía de un comando

```
> ls -lhS ~/Downloads | head -5
```

Diagrama de la estructura del comando `ls -lhS ~/Downloads | head -5`:

- `ls`: el comando
- `-lhS`: opciones: `-l` largo, `-h` legible, `-S` por tamaño
- `~/Downloads`: argumento: sobre qué actúa
- `|`: tubería: la salida de uno entra en el otro
- `head -5`: argumento del segundo comando

Leer el manual sin morir en el intento

Cada comando tiene su página de manual. Dentro de `man` navegas con `Espacio` (avanzar), `b` (retroceder), `/` (buscar), `n` (siguiente resultado) y `q` (salir).

```
> man diskutil          # manual completo
> man -k password       # busca comandos relacionados con "password"
> diskutil              # muchos comandos de Apple listan sus verbos sin
argumentos
> type ls; which -a python3 # qué se ejecuta realmente y desde dónde
```

TRUCO

¿Quieres leer un manual con formato bonito? `man -t grep | open -f -a Preview` lo convierte a PostScript y lo abre en Vista Previa. En versiones recientes de macOS en las que Vista Previa no abra PostScript, usa `mandoc -T pdf "$(man -w grep)" > grep.pdf && open grep.pdf`.

BSD frente a GNU

Las utilidades de macOS vienen de BSD, no de GNU. En la práctica, eso significa pequeñas diferencias que confunden a quien llega de Linux:

Tarea	macOS (BSD)	Linux (GNU)
Editar en sitio con sed	<code>sed -i '' 's/a/b/' f</code>	<code>sed -i 's/a/b/' f</code>
Fecha de ayer	<code>date -v-1d</code>	<code>date -d yesterday</code>
Tamaño de archivo	<code>stat -f %z f</code>	<code>stat -c %s f</code>
Base64 decodificar	<code>base64 -d</code> (o <code>-D</code> en versiones antiguas)	<code>base64 -d</code>
Ordenar por versión	<code>sort -V</code> (macOS reciente)	<code>sort -V</code>

Si necesitas las versiones GNU, Homebrew las instala con prefijo `g`: `brew install coreutils gnu-sed` te da `gsed`, `gdate`, `gls`, etc. Lo veremos en el capítulo 12.

zsh a fondo

Escribir menos y hacer más: atajos, historial y las expansiones que separan al aficionado del experto.

Atajos de edición de línea

zsh usa por defecto atajos estilo Emacs. Memorizar seis o siete te ahorra horas al año.

Atajo	Acción
<code>Ctrl + A</code> / <code>Ctrl + E</code>	Ir al inicio / final de la línea
<code>⌘ + ←</code> / <code>⌘ + →</code>	Saltar palabra (en Terminal, activa «Usar Opción como tecla Meta»)
<code>Ctrl + W</code>	Borrar la palabra anterior
<code>Ctrl + U</code> / <code>Ctrl + K</code>	Borrar hasta el inicio / hasta el final
<code>Ctrl + Y</code>	Pegar lo último borrado
<code>Ctrl + R</code>	Búsqueda inversa en el historial
<code>Ctrl + X</code> <code>Ctrl + E</code>	Abrir la línea actual en tu editor (requiere configurarlo, ver cap. 12)
<code>Ctrl + L</code> / <code>⌘ + K</code>	Limpiar pantalla / limpiar también el scroll
<code>⌘ + clic</code>	En Terminal.app, mueve el cursor a esa posición

El historial como superpoder

```
> sudo !!           # repite el último comando con sudo
> cd !$            # último argumento del comando anterior
> ^jpg^png         # repite el último comando cambiando jpg por png
> !git             # ejecuta el último comando que empezaba por git
> !git:p           # ...pero solo lo muestra, sin ejecutarlo
> history -10       # últimos 10 comandos
> history -E 1 | grep ssh # historial completo con fecha y hora
> fc               # edita el último comando en tu editor y lo ejecuta
```

Si escribes `!$` y pulsas `Espacio`, zsh lo expande en pantalla antes de ejecutarlo, para que veas qué va a pasar. Para aprovechar el historial entre ventanas, añade a tu `~/ .zshrc` :

```
HISTSIZE=100000
SAVEHIST=100000
setopt SHARE_HISTORY HIST_IGNORE_ALL_DUPS HIST_IGNORE_SPACE
EXTENDED_HISTORY
```

TRUCO

Con `HIST_IGNORE_SPACE` , cualquier comando que empiece por un espacio no se guarda en el historial. Ideal para comandos que contienen contraseñas o tokens.

Expansión de llaves

```
> mkdir -p proyecto/{src,docs,tests}/{v1,v2} # crea 6 carpetas de golpe
> cp config.yml{,.bak}                       # = cp config.yml
config.yml.bak
> mv informe.{txt,md}                        # renombra cambiando
extensión
> touch capitulo_{01..12}.md                 # con ceros a la izquierda
> echo {a..z..3}                             # a d g j m p s v y
```

Globbering extendido de zsh

Aquí zsh brilla por encima de bash. `**` busca recursivamente y los **calificadores** entre paréntesis filtran por tipo, tamaño, fecha o permisos, sin necesidad de `find` .

```

> ls **/*.pdf                # todos los PDF en cualquier subcarpeta
> ls **/*(.)                 # solo archivos regulares
> ls **/*(/)                 # solo directorios
> ls -lh **/*(.Lm+100)      # archivos de más de 100 MB
> ls *(.mh-2)                # modificados en las últimas 2 horas
> ls *(.md+30)               # modificados hace más de 30 días
> ls *(.om[1,5])             # los 5 más recientes
> rm -i **/*.DS_Store(.N)    # N: no da error si no hay coincidencias

```

Activa además el globbing extendido para negaciones y exclusiones:

```

> setopt EXTENDED_GLOB
> ls ^*.log                  # todo lo que NO acabe en .log
> ls **/*.js~**/node_modules/** # .js excepto los de node_modules
> ls (#i)*readme*            # sin distinguir mayúsculas

```

Calificador	Significado	Calificador	Significado
.	archivo regular	L+n	tamaño > n (k, m, g)
/	directorio	m-n	modificado hace menos de n días
@	enlace simbólico	mh , mm	en horas, en minutos
*	ejecutable	om / Om	ordenar por fecha ↓ / ↑
N	sin error si vacío	[1,5]	rango de resultados

Modificadores de rutas y variables

zsh puede descomponer rutas sin llamar a `basename` ni `dirname` :

```

> f=~/Documents/obra/presupuesto_2026.bc3
> echo ${f:t}                # presupuesto_2026.bc3 (tail)
> echo ${f:h}                # /Users/marlon/Documents/obra (head)
> echo ${f:r}                # ../presupuesto_2026 (root, sin extensión)
> echo ${f:e}                # bc3 (extensión)
> echo ${f:t:r:u}            # PRESUPUESTO_2026 (encadenados, u = mayúsculas)

> s="Hola Mundo"
> echo ${s:l} ${#s} ${s/Mundo/Mac} ${s[1,4]}
hola mundo 10 Hola Mac Hola

```

Renombrado masivo con zmv

`zmv` es una función incluida en `zsh` que renombra con patrones. Usa siempre `-n` primero para ver qué haría sin tocar nada.

```
> autoload -U zmv
> zmv -n '(*).jpeg' '$1.jpg'          # simulación
> zmv '(*).jpeg' '$1.jpg'            # ejecución real
> zmv '(*)(*)' '$1_$2'                # espacios por guiones bajos
> zmv -W 'IMG_*.HEIC' 'vacaciones_*.heic' # -W: comodines simples
> zmv '(**/)(*)' '$1${(L)2}'          # todo a minúsculas,
recursivo
```

Navegación rápida entre carpetas

```
> cd -          # vuelve a la carpeta anterior
> setopt AUTO_CD AUTO_PUSHD PUSHD_IGNORE_DUPS
> Documents     # con AUTO_CD, escribir el nombre basta para entrar
> dirs -v       # pila de carpetas visitadas, numeradas
> cd ~2         # salta a la entrada 2 de la pila
> cd /u/l/b Tab # zsh completa a /usr/local/bin
```

Redirecciones y tuberías

La filosofía Unix: programas pequeños que hacen una cosa bien y se conectan entre sí.

Los tres canales

Todo proceso tiene tres canales abiertos: entrada estándar (**0**, stdin), salida estándar (**1**, stdout) y salida de errores (**2**, stderr). Redirigirlos es la base de todo lo demás.

```
> ls > lista.txt           # stdout a archivo (sobrescribe)
> ls >> lista.txt          # añade al final
> find / -name x 2>/dev/null # descarta los errores de permisos
> make > build.log 2>&1    # stdout y stderr al mismo archivo
> make &> build.log        # atajo equivalente en zsh
> make 2>&1 | less         # errores también por la tubería
> make |& less             # atajo zsh de lo anterior
> wc -l < datos.csv        # archivo como entrada
```

CUIDADO

Por defecto, `>` sobrescribe el archivo sin preguntar. Para protegerte de accidentes añade `setopt NO_CLOBBER` a tu `.zshrc`; cuando de verdad quieras sobrescribir, usa `>|`.

Multios: escribir en varios sitios

zsh permite varias redirecciones de salida a la vez sin necesidad de `tee`:

```
> date > a.txt > b.txt      # escribe en ambos archivos
> ls > lista.txt | grep pdf  # guarda y a la vez filtra
```

En scripts portables es mejor usar `tee`, que funciona en cualquier shell:

```
> brew upgrade 2>&1 | tee -a ~/logs/brew.log
> echo "127.0.0.1 test.local" | sudo tee -a /etc/hosts # sudo con
redirección
```

POR QUÉ TEE CON SUDO

sudo echo x >> /etc/hosts falla: la redirección la hace tu shell (sin privilegios), no sudo. Con | sudo tee -a es tee quien escribe, y sí tiene permisos.

Here-docs y here-strings

```
> cat <<EOF > config.ini
[servidor]
host=$HOST
fecha=$(date +%F)
EOF

> cat <<'EOF' > script.sh # con comillas: NO expande variables
echo "Usuario: $USER"
EOF

> grep -c a <<< "banana" # here-string: texto como stdin
```

Sustitución de comandos y de procesos

```
> echo "Hoy es $(date +%A) y tengo $(ls | wc -l | tr -d ' ') archivos"
> diff <(ls carpetaA) <(ls carpetaB) # compara dos listados sin
archivos temporales
> comm -13 <(sort viejos.txt) <(sort nuevos.txt) # líneas solo en nuevos
> open =(curl -s https://example.com) # zsh: =( ) crea un archivo
temporal real
```

<(...) entrega una tubería con nombre; algunos programas necesitan un archivo de verdad que puedan releer, y para eso zsh ofrece =(...) .

Operadores de control

```
> brew update && brew upgrade           # el segundo solo si el primero tuvo
éxito
> ping -c1 nas.local || say "NAS caído"  # el segundo solo si el primero
falla
> cd /tmp; ls                             # secuencial, sin condiciones
> { date; uptime; } > estado.txt          # agrupa salidas
> (cd ~/proyecto && make)                 # subshell: no cambia tu carpeta
actual
> echo $?                                # código de salida del último
comando (0 = OK)
```

xargs: de líneas a argumentos

`xargs` convierte la salida de un comando en argumentos de otro. Combinado con separadores nulos es seguro con nombres que contienen espacios.

```
> find . -name '*.tmp' -print0 | xargs -0 rm
> cat urls.txt | xargs -n1 -P4 curl -s0    # 4 descargas en paralelo
> ls *.heic | xargs -I{} sips -s format jpeg {} --out {}.jpg
> mdfind -0 -name factura | xargs -0 ls -lh
```

Trabajos en segundo plano

```
> ./largo.sh &                          # lanza en segundo plano
>           # suspende el proceso en primer plano
> bg ; fg ; jobs                         # reanudar detrás, traer delante, listar
> nohup ./servidor.sh > srv.log 2>&1 &   # sobrevive al cerrar la ventana
> disown %1                             # desvincula un trabajo de la shell
> wait                                  # espera a que terminen todos
```

Buscar archivos y contenido

find recorre el disco; mdfind consulta el índice de Spotlight. Saber cuándo usar cada uno marca la diferencia.

find: el clásico infalible

```
> find ~ -name '*.pdf' -size +10M # PDF de más de 10 MB
> find . -type f -mtime -1 # modificados en las
últimas 24 h
> find . -type f -mmin -30 # en los últimos 30
minutos
> find . -iname '*factura*' -not -path '*/node_modules/*'
> find . -type d -empty -delete # borra carpetas vacías
> find . -name '*.log' -mtime +30 -exec rm {} + # logs de más de un mes
> find . -type f -perm +111 # ejecutables (sintaxis
BSD)
> find . -newer referencia.txt # más nuevos que un
archivo
> find /Applications -maxdepth 1 -name '*.app' | wc -l
```

La diferencia entre `-exec ... \;` y `-exec ... +` es importante: el primero lanza el comando una vez por archivo; el segundo agrupa muchos archivos en una sola llamada, mucho más rápido.

mdfind: Spotlight desde la consola

Spotlight indexa no solo nombres, sino contenido y metadatos (autor, dimensiones de imagen, duración de audio, etiquetas de color...). `mdfind` responde en milisegundos donde `find` tardaría minutos.

```

> mdfind "presupuesto hormigón" # busca en contenido y
nombre
> mdfind -name contrato # solo por nombre
> mdfind -onlyin ~/Documents "kMDItemContentType == 'com.adobe.pdf'"
> mdfind "kMDItemFSSize > 1000000000" # archivos de más de 1 GB
> mdfind "kMDItemContentModificationDate >= \$time.today(-7)"
> mdfind "kMDItemPixelWidth >= 4000" # imágenes de 4000 px de
ancho o más
> mdfind "kMDItemUserTags == 'Rojo'" # con etiqueta de color
Rojo
> mdfind "kMDItemWhereFroms == '*github.com*'" # descargados desde
GitHub
> mdfind -live -name '*.dmg' # se queda escuchando
nuevos resultados
> mdfind -count -onlyin ~ "kind:pdf" # sintaxis abreviada
estilo Spotlight

```

¿Qué atributos tiene un archivo? mdls

```

> mdls foto.heic
> mdls -name kMDItemWhereFroms -name kMDItemDownloadedDate archivo.zip
> mdls -raw -name kMDItemDurationSeconds podcast.mp3

```

Con `kMDItemWhereFroms` descubres desde qué web descargaste algo, aunque haga meses.

Mantener el índice: mdutil

```

> mdutil -s / # estado de indexación del volumen
> sudo mdutil -E / # borra y regenera el índice (si
Spotlight falla)
> sudo mdutil -i off /Volumes/Backup # desactiva indexado en un disco
externo

```

grep: buscar dentro de los archivos

```
> grep -rn "TODO" src/                # recursivo con número de línea
> grep -ril "password" . 2>/dev/null  # solo nombres de archivo, sin
mayúsculas
> grep -E "^(ERROR|WARN)" app.log     # expresiones regulares
extendidas
> grep -v "^#" config.conf | grep -v "^$" # sin comentarios ni líneas
vacías
> grep -C3 "Exception" app.log        # 3 líneas de contexto antes y
después
> grep -o "[0-9]\{1,3\}\(\.[0-9]\{1,3\}\)\{3\}" access.log | sort | uniq -
c | sort -rn
> grep -c "" archivo.txt              # cuenta líneas (incluida la
última sin \n)
```

Las alternativas modernas: ripgrep y fd

`ripgrep` (`rg`) y `fd` son reescrituras modernas de `grep` y `find`: muchísimo más rápidas, respetan `.gitignore` y tienen una sintaxis más amable. Se instalan con Homebrew.

```
> brew install ripgrep fd
> rg "def calcular" -t py              # solo en archivos Python
> rg -i "iva" --glob '!*.min.js'
> rg -l "odoo" | xargs wc -l
> fd factura ~/Documents              # búsqueda por nombre, sin sintaxis
-name
> fd -e pdf -S +5m                    # PDF de más de 5 MB
> fd -e heic -x sips -s format jpeg {} --out {}.jpg # ejecuta por cada
resultado
```

Procesar texto como un profesional

Logs, CSV, JSON, listados: en Unix todo es texto, y el texto se transforma con unas pocas herramientas.

Cortar, ordenar y contar

```
> cut -d';' -f1,3 clientes.csv           # columnas 1 y 3, separador ;
> sort -t';' -k3 -n clientes.csv         # ordena numéricamente por la
columna 3
> sort -u nombres.txt                   # ordena y elimina
duplicados
> sort archivo | uniq -c | sort -rn | head # el "top 10" universal
> tr '[:lower:]' '[:upper:]' < a.txt      # a mayúsculas
> tr -d '\r' < windows.txt > unix.txt    # quita retornos de carro de
Windows
> paste -sd+ numeros.txt | bc            # suma una columna de
números
> column -t -s';' clientes.csv | less -S # CSV alineado en columnas
> wc -lwc informe.txt                   # líneas, palabras y bytes
> head -n 20 / tail -n 20 / tail -f app.log # principio, final, seguir en
vivo
```

sed: editar en flujo

```
> sed 's/http:/https:/g' urls.txt        # sustitución global
> sed -i '' 's/2025/2026/g' *.md          # en sitio (ojo: '')
obligatorio en macOS)
> sed -i '.bak' 's/foo/bar/' config.txt   # en sitio guardando copia
.bak
> sed -n '10,20p' archivo.txt             # imprime solo las líneas 10
a 20
> sed '/^$/d' archivo.txt                 # borra líneas vacías
> sed -E 's/([0-9]{2})\([0-9]{2})\([0-9]{4})/\3-\2-\1/g' fechas.txt #
dd/mm/aaaa → aaaa-mm-dd
> sed '1d' datos.csv                      # quita la cabecera
```

awk: un lenguaje en una línea

awk divide cada línea en campos (`$1` , `$2` ...) y permite cálculos, condiciones y acumuladores. Es la navaja suiza para datos tabulares.

```
> awk '{print $1, $NF}' archivo # primera y última columna
> awk -F';' '$4 > 1000 {print $2}' ventas.csv # filtra por valor
> awk -F';' '{s+=$4} END {printf "%.2f €\n", s}' ventas.csv # suma
> awk -F';' '{t[$1]+=$4} END {for(k in t) print k, t[k]}' ventas.csv #
total por cliente
> awk 'NR%2==0' archivo # solo líneas pares
> awk '!visto[$0]++' archivo # quita duplicados SIN
reordenar
> awk 'length > 120' codigo.py # líneas de más de 120
caracteres
> ps aux | awk '{m+=$6} END {print m/1024 " MB"}' # RAM total usada por
procesos
```

JSON con jq

Las APIs hablan JSON. `jq` lo filtra, transforma y formatea. En macOS Sequoia y posteriores viene incluido en el sistema; en versiones anteriores, `brew install jq`.

```
> curl -s https://api.github.com/repos/odoo/odoo | jq .
> ... | jq '.stargazers_count'
> ... | jq -r '.[.] | "\(.name)\t\(.size)"' # -r: texto
sin comillas
> jq '[.] | select(.precio > 100)] | length' productos.json
> jq 'map({id, nombre: .name})' datos.json # reestructurar
> jq -r '.[.] | [.id,.name,.price] | @csv' p.json > p.csv # JSON → CSV
> jq -s 'add' partel.json parte2.json # fusionar
arrays
```

plutil: JSON y plist al estilo Apple

Las preferencias de macOS se guardan en archivos `.plist` (XML o binario). `plutil` los lee, valida y convierte, y además sabe extraer valores de JSON sin `jq`.

```

> plutil -p ~/Library/Preferences/com.apple.dock.plist      # vista legible
> plutil -convert xml1 -o - archivo.plist                  # binario → XML
por pantalla
> plutil -convert json -o datos.json archivo.plist
> plutil -lint Info.plist                                  # valida la
sintaxis
> plutil -extract CFBundleShortVersionString raw
/Applications/Safari.app/Contents/Info.plist
> echo '{"a":{"b":42}}' | plutil -extract a.b raw -         # extraer de
JSON por stdin

```

VERSIÓN DE CUALQUIER APP

El truco de `plutil -extract CFBundleShortVersionString raw` sobre el `Info.plist` de una app te da su versión sin abrirla. Muy útil en scripts de inventario.

Pequeñas joyas

```

> iconv -f ISO-8859-1 -t UTF-8 viejo.csv > nuevo.csv      # arreglar tildes
rotas
> file -I archivo.csv                                     # detecta tipo y
codificación
> shasum -a 256 instalador.dmg                             # verificar
descargas
> md5 -q archivo                                           # hash MD5 (versión
macOS)
> base64 -i imagen.png | pbcopy                             # imagen a base64 en
el portapapeles
> split -l 1000 grande.csv parte_                          # trocear en
archivos de 1000 líneas
> nl -ba script.sh                                          # numerar todas las
líneas
> rev <<< "consola"                                         # alosnoc

```

Los comandos exclusivos de macOS

Las utilidades que Apple añadió encima de Unix y que conectan la consola con el escritorio.

open: el puente con el Finder

```
> open . # abre la carpeta actual en el Finder
> open informe.pdf # con la app predeterminada
> open -a "Visual Studio Code" . # con una app concreta
> open -R ~/Downloads/x.zip # muestra el archivo seleccionado en
el Finder
> open -e notas.txt # con TextEdit
> open -t notas.txt # con el editor de texto
predeterminado
> open https://www.odoo.com # URL en el navegador predeterminado
> open -na Safari # fuerza una NUEVA instancia
> open -g -a Music # abre en segundo plano, sin robar el
foco
> ls -la | open -f # salida de un comando en TextEdit
> open "x-apple.systempreferences:com.apple.preference.security" # panel
de Privacidad
```

pbcopy y pbpaste: el portapapeles

```
> cat ~/.ssh/id_ed25519.pub | pbcopy # copiar tu clave pública
> pbpaste | sort -u | pbcopy # ordenar y deduplicar lo
copiado
> pbpaste | jq . | pbcopy # formatear el JSON que
tienes copiado
> pbpaste | wc -w # contar palabras del
portapapeles
> pbpaste > borrador.md # volcar a archivo
> pwd | tr -d '\n' | pbcopy # copiar la ruta actual sin
salto de línea
```

say: tu Mac habla

```
> say "Compilación terminada"
> say -v '?' | grep es_          # voces disponibles en
español
> say -v Mónica -r 180 "Hola Marlon"    # voz y velocidad
> say -f capitulo.txt -o capitulo.aiff   # texto a archivo de audio
> ./backup.sh && say "Listo" || say "Error" # aviso sonoro al acabar
```

caffeinate: que no se duerma

```
> caffeinate          # no dormir mientras esté abierto (Ctrl+C
para salir)
> caffeinate -d -t 3600    # pantalla encendida una hora
> caffeinate -dims        # ni pantalla, ni reposo, ni disco
> caffeinate -i rsync -a origen/ destino/  # despierto solo mientras dure
el comando
> caffeinate -w 4521      # mientras viva el proceso con PID 4521
```

screencapture: capturas avanzadas

```
> screencapture pantalla.png          # pantalla completa
> screencapture -i zona.png           # selección interactiva
> screencapture -ic                   # selección directa al
portapapeles
> screencapture -T 5 retardo.png      # con 5 segundos de retardo
> screencapture -x -t jpg silenciosa.jpg # sin sonido y en JPG
> screencapture -v video.mov          # grabación de vídeo de la
pantalla
> screencapture -l$(osascript -e 'tell app "Safari" to id of window 1')
safari.png
```

sips: edición de imágenes por lotes

sips (Scriptable Image Processing System) viene de serie y es perfecto para procesar cientos de fotos sin abrir ninguna app.

```

> sips -g pixelWidth -g pixelHeight foto.jpg      # consultar dimensiones
> sips -Z 1600 *.jpg                               # lado mayor a 1600 px
(;sobrescribe!)
> mkdir web && sips -Z 1200 *.jpg --out web/       # redimensiona a otra
carpeta
> sips -s format jpeg foto.heic --out foto.jpg    # HEIC → JPG
> sips -s format jpeg -s formatOptions 70 a.png --out a.jpg # calidad 70
> sips -r 90 escaneo.png                          # rotar 90°
> sips -c 800 800 foto.jpg                        # recortar al centro
800x800
> for f in *.heic; do sips -s format jpeg "$f" --out "${f:r}.jpg"; done

```

textutil: convertir documentos

```

> textutil -convert docx notas.rtf
> textutil -convert txt contrato.docx -output contrato.txt
> textutil -convert html *.docx                # lote
> textutil -cat docx cap*.docx -output libro.docx # unir varios en uno
> textutil -info informe.docx                  # palabras, autor,
fechas

```

Más utilidades que pocos conocen

```

> qlmanage -p plano.pdf                        # Vista Rápida desde la consola
> qlmanage -t -s 512 -o . video.mov           # genera miniatura PNG
> afplay /System/Library/Sounds/Glass.aiff    # reproducir audio
> ainfo cancion.m4a                           # info técnica de un audio
> afconvert -f m4af -d aac voz.wav voz.m4a    # convertir audio
> sw_vers                                     # versión de macOS
> system_profiler SPHardwareDataType           # modelo, chip, RAM, número de
serie
> system_profiler SPPowerDataType | grep -i cycle # ciclos de la batería
> sysctl -n machdep.cpu.brand_string          # nombre del procesador
> sysctl -n hw.memsize | awk '{print $1/1073741824 " GB"}'
> pmset -g batt                               # estado de batería
> pmset -g assertions                         # qué está impidiendo que el Mac
duerma
> pmset sleepnow                             # dormir ahora
> sudo shutdown -r +10                       # reiniciar en 10 minutos
> softwareupdate -l                           # actualizaciones disponibles
> sudo softwareupdate -ia                     # instalarlas todas
> softwareupdate --list-full-installers       # instaladores completos de macOS

```

defaults: los ajustes ocultos

Cientos de preferencias que no aparecen en Ajustes del Sistema están a un comando de distancia.

Cada app guarda sus preferencias en un **dominio**, normalmente con forma de identificador inverso (`com.apple.finder`, `com.apple.dock`). El comando `defaults` lee y escribe esos dominios. Tras cambiar algo, suele hacer falta reiniciar la app afectada con `killall`.

Leer antes de escribir

```
> defaults domains | tr ',' '\n' | grep apple | head # dominios
existentes
> defaults read com.apple.dock # todo el dominio
> defaults read com.apple.dock autohide # una clave
concreta
> defaults read-type com.apple.dock tileSize # tipo de dato
> defaults find screenshot # busca una
palabra en todos los dominios
```

TÉCNICA DEL DETECTIVE

¿No sabes qué clave controla un ajuste? Vuelca el dominio, cambia el ajuste en la interfaz gráfica y compara: `defaults read com.apple.finder > antes.txt`, cambias la opción, `defaults read com.apple.finder > despues.txt` y finalmente `diff antes.txt despues.txt`.

Finder

```
> defaults write com.apple.finder AppleShowAllFiles -bool true #  
archivos ocultos  
> defaults write NSGlobalDomain AppleShowAllExtensions -bool true # todas  
las extensiones  
> defaults write com.apple.finder ShowPathbar -bool true # barra  
de ruta  
> defaults write com.apple.finder ShowStatusBar -bool true # barra  
de estado  
> defaults write com.apple.finder _FXShowPosixPathInTitle -bool true #  
ruta completa en el título  
> defaults write com.apple.finder FXPreferredViewStyle -string "Nlsv" #  
vista de lista por defecto  
> defaults write com.apple.finder _FXSortFoldersFirst -bool true #  
carpetas primero  
> defaults write com.apple.desktopservices DSDontWriteNetworkStores -bool  
true # sin .DS_Store en red  
> killall Finder
```

Atajo gráfico equivalente para los ocultos:  +  +  en cualquier ventana del Finder.

Dock

```
> defaults write com.apple.dock autohide -bool true  
> defaults write com.apple.dock autohide-delay -float 0 #  
aparece sin espera  
> defaults write com.apple.dock autohide-time-modifier -float 0.3 #  
animación rápida  
> defaults write com.apple.dock tilesize -int 42  
> defaults write com.apple.dock show-recents -bool false  
> defaults write com.apple.dock static-only -bool true # solo  
apps abiertas  
> defaults write com.apple.dock persistent-apps -array-add '{"tile-  
type"="spacer-tile";}' # separador  
> killall Dock
```

Capturas de pantalla

```
> mkdir -p ~/Pictures/Capturas
> defaults write com.apple.screencapture location ~/Pictures/Capturas
> defaults write com.apple.screencapture type jpg
> defaults write com.apple.screencapture disable-shadow -bool true
> defaults write com.apple.screencapture name "Captura"
> killall SystemUIServer
```

Teclado y sistema

```
> defaults write NSGlobalDomain KeyRepeat -int 2 #
repetición muy rápida
> defaults write NSGlobalDomain InitialKeyRepeat -int 15
> defaults write NSGlobalDomain ApplePressAndHoldEnabled -bool false #
repetir en vez de acentos
> defaults write NSGlobalDomain NSAutomaticSpellingCorrectionEnabled -bool
false
> defaults write NSGlobalDomain NSDocumentSaveNewDocumentsToCloud -bool
false # guardar en local
> defaults write com.apple.LaunchServices LSQuarantine -bool false # sin
aviso de "descargado de Internet"*
```

* Desactivar avisos de cuarentena reduce la seguridad. Valóralo con cuidado.

Deshacer, respaldar y restaurar

```
> defaults delete com.apple.dock autohide-delay # vuelve al valor
por defecto
> defaults export com.apple.dock ~/dock_backup.plist # exportar dominio
entero
> defaults import com.apple.dock ~/dock_backup.plist # restaurarlo
> killall Dock
```

IMPORTANTE

No borres dominios enteros (`defaults delete com.apple.finder`) salvo que sepas lo que haces: perderás todas las preferencias de esa app. Exporta siempre antes de experimentar. Además, Apple renombra o elimina claves entre versiones; si algo no surte efecto, puede que ya no exista.

Procesos, memoria y rendimiento

Qué está consumiendo tu Mac, quién tiene abierto ese archivo y por qué el ventilador no para.

Listar y encontrar procesos

```
> ps aux                                # todos los procesos
> ps aux | sort -nrk3 | head -5         # top 5 por CPU
> ps aux | sort -nrk4 | head -5         # top 5 por memoria
> ps -axo pid,ppid,%cpu,%mem,etime,comm # columnas a medida
> pgrep -fl python                      # PID y comando de procesos
python
> pstree                                # árbol (brew install pstree)
```

top en modo experto

```
> top -o cpu                            # ordenado por CPU
> top -o mem -n 10                      # 10 procesos con más memoria
> top -l 1 -s 0 | head -12              # una sola muestra, ideal para scripts
> top -pid 4521                          # vigilar un proceso concreto
```

Alternativas visuales: `htop` y `btop` (Homebrew), con colores, ratón y árbol de procesos.

Terminar procesos

```
> kill 4521                             # señal TERM: pide cierre ordenado
> kill -9 4521                           # señal KILL: cierre forzoso, último recurso
> pkill -f "node server"                 # por patrón del comando
> killall Safari                         # por nombre
> killall -STOP Spotify ; killall -CONT Spotify # congelar y descongelar
> renice +10 -p 4521                     # bajar prioridad
> nice -n 15 ./tarea.sh                  # lanzar ya con prioridad baja
```

Isof: quién tiene qué abierto

```
> lsof -i :8069 # qué proceso usa el puerto 8069 (¡hola,
Odoos!)
> lsof -nP -iTCP -sTCP:LISTEN # todos los puertos en escucha
> lsof /Volumes/USB # por qué no puedo expulsar el disco
> lsof -p 4521 # archivos abiertos por un proceso
> lsof +D ~/proyecto # procesos con archivos abiertos en una
carpeta
> kill $(lsof -ti :3000) # matar lo que ocupa el puerto 3000
```

Memoria

```
> vm_stat # estadísticas de memoria virtual (en
páginas)
> memory_pressure # nivel de presión de memoria del sistema
> sysctl vm.swapusage # uso de swap
> sudo purge # vacía cachés de disco (para pruebas de
rendimiento)
> footprint -p Safari # huella de memoria real de un proceso
```

Diagnóstico profundo

```
> sudo fs_usage -w -f filesystems Finder # actividad de disco de un
proceso en vivo
> sudo fs_usage -w -f network # actividad de red a nivel de
llamadas
> iostat -w 2 # E/S de disco cada 2 segundos
> sudo powermetrics --samplers cpu_power,gpu_power -i 1000 -n 5 #
consumo en vatios
> sudo powermetrics --samplers smc -n 1 # temperaturas y ventiladores
(Intel)
> sample Safari 5 -file safari.txt # muestreo de pila 5 s (app
colgada)
> spindump # informe de cuelgues de todo el
sistema
> nettop -m route # tráfico de red por proceso
```

El registro unificado: log

Desde macOS Sierra, casi todo el sistema escribe en el **registro unificado**. La app Consola lo muestra, pero `log` es mucho más potente.

```
> log show --last 10m --predicate 'eventMessage contains "error"'
> log show --last 1h --predicate 'process == "kernel"' --style compact
> log show --last 2d --predicate 'eventMessage contains "Wake reason"' #
por qué despertó
> log stream --predicate 'subsystem == "com.apple.bluetooth"' --level
debug
> log stream --process Finder # en vivo, solo el Finder
> log show --last boot | head # desde el último arranque
> last reboot | head -5 ; uptime # historial de reinicios y tiempo
encendido
```

Discos, archivos y copias

Particiones, imágenes de disco, copias incrementales y los metadatos invisibles de APFS.

Espacio en disco

```
> df -h / # espacio libre
> du -sh ~/Library/Caches # tamaño de una carpeta
> du -hd1 ~ 2>/dev/null | sort -hr | head # las carpetas más grandes de
tu home
> du -sh ~/Library/Developer/* 2>/dev/null | sort -hr # Xcode y
simuladores: grandes culpables
> brew install ncdu && ncdu ~ # explorador interactivo de
espacio
```

diskutil: gestión de discos

```
> diskutil list # todos los discos y
particiones
> diskutil info / # detalles del volumen de
arranque
> diskutil apfs list # contenedores y volúmenes APFS
> diskutil verifyVolume / # comprobar integridad
> diskutil eject disk4 # expulsar
> diskutil unmount force /Volumes/USB # desmontar a la fuerza
> diskutil eraseDisk ExFAT PENDRIVE GPT disk4 # formatear un USB (¡borra
todo!)
> diskutil apfs addVolume disk3 APFS Proyectos # nuevo volumen que
comparte espacio
```

VERIFICA EL IDENTIFICADOR

Antes de cualquier `erase`, ejecuta `diskutil list` y confirma dos veces que `disk4` es el pendrive y no tu disco interno o tu copia de seguridad. No hay papeleta para esto.

hdiutil: imágenes de disco

```
> hdiutil create -volname Entrega -srcfolder ./entrega -ov -format UDZO entrega.dmg
> hdiutil create -encryption AES-256 -size 2g -fs APFS -volname Privado privado.dmg
> hdiutil attach instalador.dmg # montar
> hdiutil detach /Volumes/Entrega # desmontar
> hdiutil makehybrid -iso -joliet -o disco.iso carpeta/ # crear ISO
```

Copiar bien: cp, ditto y rsync

```
> cp -c grande.mov copia.mov # clon APFS: instantáneo y sin ocupar espacio extra
> ditto origen/ destino/ # copia conservando metadatos, permisos y ACL
> ditto -c -k --keepParent Carpeta Carpeta.zip # zip idéntico al de "Comprimir" del Finder
> ditto -x -k archivo.zip destino/ # descomprimir
> rsync -avh --progress origen/ destino/ # sincronización incremental
> rsync -avh --delete origen/ destino/ # espejo exacto (borra lo sobrante)
> rsync -avhn --delete origen/ destino/ # -n: simulación, nada se toca
> rsync -avz -e ssh proyecto/ usuario@servidor:/srv/proyecto/
```

Fíjate en la barra final de `origen/`: con ella se copia el *contenido*; sin ella, la carpeta misma dentro de destino.

Compresión

```
> tar -czf copia_$(date +%F).tar.gz proyecto/
> tar -xzf copia.tar.gz -C /destino
> tar -tzf copia.tar.gz | head # listar sin extraer
> zip -er secreto.zip carpeta/ # zip con contraseña
> aa archive -d carpeta -o carpeta.aar # Apple Archive, formato nativo moderno
```

Atributos extendidos y banderas

macOS adjunta metadatos invisibles a los archivos: de dónde vienen, si están en cuarentena, etiquetas de color... Los gestiona `xattr`.

```

> ls -l@ archivo.dmg # ver atributos extendidos
> xattr -l archivo.dmg # con sus valores
> xattr -d com.apple.quarantine App.app # quitar cuarentena (solo si
confías)
> xattr -cr carpeta/ # limpiar todos, recursivo
> chflags hidden carpeta # ocultar en el Finder
> chflags nohidden ~/Library # mostrar la Biblioteca
> chflags uchg contrato.pdf # bloquear (el "Bloqueado" del
Finder)
> SetFile -d "01/15/2026 10:00:00" f.txt # cambiar fecha de creación
(con Xcode CLT)

```

Time Machine desde la consola

```

> tmutil status # ¿está copiando ahora?
> tmutil startbackup --auto # lanzar copia
> tmutil latestbackup # última copia
> tmutil listbackups # todas las copias
> sudo tmutil addexclusion -p ~/Downloads # excluir carpeta
> tmutil listlocalsnapshots / # instantáneas locales APFS
> tmutil thinlocalsnapshots / 10000000000 4 # liberar ~10 GB de
instantáneas

```

Redes desde la consola

Diagnosticar, configurar y transferir: la red se entiende mejor desde la Terminal.

¿Dónde estoy conectado?

```
> ipconfig getifaddr en0           # IP local de la interfaz
principal
> ipconfig getsummary en0          # resumen completo: DHCP,
router, DNS, SSID
> curl -s https://ifconfig.me ; echo # IP pública
> networksetup -listallhardwareports # qué es en0, en1... en tu
Mac
> route -n get default | grep gateway # puerta de enlace
> scutil --dns | grep nameserver | sort -u # servidores DNS en uso
> arp -a                          # dispositivos vistos en la
red local
```

networksetup: configurar sin Ajustes

```
> networksetup -setairportpower en0 off && networksetup -setairportpower
en0 on # reiniciar Wi-Fi
> networksetup -setdnsservers Wi-Fi 1.1.1.1 9.9.9.9 # cambiar DNS
> networksetup -setdnsservers Wi-Fi Empty           # volver al DNS
automático
> networksetup -listpreferredwirelessnetworks en0   # redes Wi-Fi
guardadas
> networksetup -setwebproxy Wi-Fi proxy.local 8080
> sudo dscacheutil -flushcache; sudo killall -HUP mDNSResponder # vaciar
caché DNS
```

CONTRASEÑA DEL WI-FI GUARDADA

`security find-generic-password -wa "NombreDeTuRed"` te muestra la contraseña de una red guardada, tras pedirte autenticación. Útil para compartirla con un invitado.

Nombre del equipo

```
> scutil --get ComputerName
> sudo scutil --set ComputerName "MacBook-Marlon"
> sudo scutil --set LocalHostName "macbook-marlon"      # el nombre .local
de Bonjour
> sudo scutil --set HostName "macbook-marlon"
```

Diagnóstico

```
> networkQuality                # test de velocidad oficial de Apple
> networkQuality -v             # con detalle: latencia, capacidad de
                                # respuesta
> ping -c 5 8.8.8.8
> traceroute -n www.odoo.com
> dig +short erp-sqlite.xyz      # resolución DNS
> dig MX gmail.com +short       # registros de correo
> dig @1.1.1.1 dominio.es ANY   # preguntar a un servidor concreto
> nc -zv servidor.com 443       # ¿está abierto el puerto?
> nc -zv 192.168.1.10 20-25     # rango de puertos
> netstat -rn                   # tabla de rutas
> sudo tcpdump -i en0 -n port 53 # capturar tráfico DNS
```

curl: el cliente HTTP definitivo

```
> curl -I https://example.com      # solo cabeceras
> curl -L -o archivo.zip https://... # sigue redirecciones y
guarda
> curl -C - -O https://.../grande.iso # reanudar descarga
interrumpida
> curl -s -X POST https://api.x.com/v1/items \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $TOKEN" \
  -d '{"nombre":"Hormigón HA-25","precio":89.5}' | jq .
> curl -F "file=@plano.pdf" https://subida.ejemplo.com
> curl -w "%{http_code} %{time_total}s\n" -o /dev/null -s
https://example.com # tiempos
> curl -u usuario https://servidor/webdav/      # autenticación básica
(pide contraseña)
```

SSH: trabajo remoto

```
> ssh-keygen -t ed25519 -C "marlon@mac"           # generar clave
> ssh-copy-id usuario@servidor                     # instalarla en el
servidor
> ssh-add --apple-use-keychain ~/.ssh/id_ed25519   # guardar la frase
en el llavero
> ssh -L 8069:localhost:8069 usuario@servidor     # túnel: Odoo
remoto en tu localhost
> ssh -D 1080 usuario@servidor                    # proxy SOCKS por
SSH
> scp informe.pdf usuario@servidor:/tmp/
> ssh servidor 'df -h; uptime'                    # comando remoto
sin sesión
```

Simplifica tus conexiones con `~/.ssh/config` :

```
Host *
  UseKeychain yes
  AddKeysToAgent yes
  IdentityFile ~/.ssh/id_ed25519

Host produccion
  HostName 203.0.113.25
  User odoo
  Port 2222
  LocalForward 8069 localhost:8069
```

Ahora basta con `ssh produccion`. Y para activar el acceso remoto a tu propio Mac: `sudo systemsetup -setremotelogin on` (equivala a «Inicio de sesión remoto» en Ajustes > General > Compartir; en macOS reciente puede requerir que la Terminal tenga Acceso total al disco).

Seguridad y llavero

Contraseñas en el llavero, firmas de código, Gatekeeper, SIP y cifrado de disco.

security: el llavero desde la consola

El llavero (Keychain) es el lugar correcto para guardar secretos que tus scripts necesitan. Nunca escribas una contraseña en texto plano dentro de un script.

```
> security add-generic-password -a "$USER" -s "api-odoo" -w      # pide el
secreto de forma oculta
> security find-generic-password -a "$USER" -s "api-odoo" -w    #
recuperarlo
> security delete-generic-password -a "$USER" -s "api-odoo"
> security find-internet-password -s github.com                 #
credenciales web
> security list-keychains
> security find-certificate -a -c "Apple" -Z | grep SHA-256     #
certificados
```

Uso típico en un script:

```
TOKEN=$(security find-generic-password -a "$USER" -s "api-odoo" -w)
curl -s -H "Authorization: Bearer $TOKEN" https://mi-odoo.com/api/...
```

Touch ID para sudo

Desde macOS Sonoma existe un archivo pensado para esto que sobrevive a las actualizaciones del sistema:

```
> sudo cp /etc/pam.d/sudo_local.template /etc/pam.d/sudo_local
> sudo nano /etc/pam.d/sudo_local
# quita el # de la línea: auth sufficient pam_tid.so
```

Desde ese momento, `sudo` te pedirá la huella en lugar de la contraseña.

Gatekeeper y firmas

```
> spctl --status # ¿Gatekeeper activo?
> spctl --assess -vv /Applications/Slack.app # ¿lo aceptaría Gatekeeper?
¿notarizada?
> codesign -dv --verbose=4 /Applications/Safari.app # quién la firmó
> codesign --verify --deep --strict App.app # integridad de la firma
> codesign -d --entitlements - App.app # permisos especiales que solicita
> pkgutil --check-signature instalador.pkg
> pkgutil --pkgs | grep -v com.apple # paquetes instalados de terceros
> pkgutil --files com.empresa.app # qué archivos instaló un paquete
```

Protecciones del sistema

```
> csrutil status # SIP (protección de integridad del sistema)
> fdesetup status # FileVault: ¿está cifrado el disco?
> sudo fdesetup list # usuarios que pueden desbloquearlo
> /usr/libexec/ApplicationFirewall/socketfilterfw --getglobalstate # cortafuegos
> sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setglobalstate on
> sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setstealthmode on
```

SIP

Desactivar SIP solo es posible desde el modo de recuperación y deja el sistema expuesto. Casi ninguna necesidad real lo justifica; si un tutorial te lo pide, busca otra forma.

Privacidad (TCC)

Los permisos de cámara, micrófono, disco completo, etc., se gestionan con `tccutil`. Solo permite *restablecer*, no conceder:

```
> tccutil reset Camera # todas las apps vuelven a preguntar
> tccutil reset ScreenCapture com.empresa.app # solo para una app
```

Utilidades criptográficas

```
> openssl rand -base64 24                # contraseña aleatoria
fuerte
> openssl enc -aes-256-cbc -pbkdf2 -salt -in datos.csv -out datos.enc  #
cifrar
> openssl enc -d -aes-256-cbc -pbkdf2 -in datos.enc -out datos.csv      #
descifrar
> openssl s_client -connect odoo.com:443 </dev/null 2>/dev/null | openssl
x509 -noout -dates                # caducidad SSL
> rm -P archivo_sensible          # sobrescribe antes de
borrar*
```

* En SSD con APFS la sobrescritura no garantiza el borrado físico; la protección real es FileVault activado.

Automatización

Scripts, AppleScript, Atajos y launchd: que el Mac trabaje mientras tú no estás.

Tu primer script zsh robusto

```
#!/bin/zsh
# backup_proyectos.zsh – copia incremental con aviso
set -euo pipefail          # para en errores, variables vacías y
fallos en tuberías

ORIGEN="$HOME/Proyectos/"
DESTINO="/Volumes/Backup/Proyectos/"
LOG="$HOME/Library/Logs/backup_proyectos.log"

if [[ ! -d "$DESTINO" ]]; then
    osascript -e 'display notification "Disco de backup no conectado" with
title "Backup"'
    exit 1
fi

echo "== $(date '+%F %T') ==" >> "$LOG"
caffeinate -i rsync -ah --delete --exclude='.DS_Store' --
exclude='node_modules' \
    "$ORIGEN" "$DESTINO" >> "$LOG" 2>&1

osascript -e 'display notification "Copia terminada" with title "Backup"
sound name "Glass"'
```

```
> chmod +x backup_proyectos.zsh && ./backup_proyectos.zsh
```

Estructuras esenciales

```
# argumentos y valores por defecto
destino=${1:~/Desktop}
(( $# == 0 )) && echo "Uso: $0 carpeta" && exit 1

# bucles
for f in *.pdf(N); do echo "Procesando ${f:t}"; done
while IFS=';' read -r codigo nombre precio; do
    echo "$codigo → $precio €"
done < partidas.csv

# funciones y case
procesar() {
    case ${1:e} in
        jpg|png) sips -Z 1200 "$1" ;;
        pdf)     echo "PDF: $1" ;;
        *)       echo "Ignorado: $1" ;;
    esac
}

# arrays y diccionarios (zsh empieza a contar en 1)
apps=(Safari Mail Notes); echo $apps[1] ${#apps}
typeset -A precio=(acero 1.2 hormigon 89.5); echo $precio[acero]

# limpieza al salir, aunque haya error
tmp=$(mktemp -d); trap 'rm -rf "$tmp"' EXIT
```

osascript: AppleScript y JavaScript desde la consola

Con `osascript` controlas apps gráficas desde la Terminal: notificaciones, diálogos, volumen, Finder, Mail...

```

> osascript -e 'display notification "Exportación lista" with title "ERP"
sound name "Ping"'
> osascript -e 'display dialog "¿Continuar?" buttons {"No","Sí"} default
button "Sí"'
> osascript -e 'set volume output volume 30'
> osascript -e 'tell application "Finder" to empty trash'
> osascript -e 'tell application "System Events" to keystroke "v" using
command down'
> osascript -e 'tell app "Finder" to get POSIX path of (target of front
window as alias)'
> osascript -e 'tell application "System Events" to tell appearance
preferences to set dark mode to not dark mode'
> osascript -l JavaScript -e
'Application("Safari").windows[0].currentTab.url()' # JXA
> nombre=$(osascript -e 'text returned of (display dialog "Nombre del
cliente:" default answer "")')

```

La primera vez que un script controla otra app, macOS pedirá permiso de Automatización o Accesibilidad para la Terminal.

Atajos (Shortcuts) desde la Terminal

```

> shortcuts list # tus atajos
> shortcuts run "Modo concentración"
> shortcuts run "Redimensionar" -i foto.jpg -o pequeña.jpg # con entrada
y salida
> echo "texto" | shortcuts run "Traducir" | pbcopy

```

Y al revés: la acción «Ejecutar script de shell» de Atajos lanza tus scripts zsh, de modo que puedes llamarlos desde la barra de menús, Siri o el Apple Watch.

launchd: el programador de tareas de macOS

En macOS, la forma nativa de ejecutar tareas periódicas o al iniciar sesión es **launchd**. Un *Launch Agent* de usuario es un archivo plist en `~/Library/LaunchAgents`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0"><dict>
  <key>Label</key>          <string>com.marlon.backup</string>
  <key>ProgramArguments</key>
  <array><string>/Users/marlon/bin/backup_proyectos.zsh</string></array>
  <key>StartCalendarInterval</key>
  <dict><key>Hour</key><integer>21</integer><key>Minute</key>
<integer>30</integer></dict>
  <key>StandardOutPath</key> <string>/tmp/backup.out</string>
  <key>StandardErrorPath</key><string>/tmp/backup.err</string>
</dict></plist>
```

```
> plutil -lint ~/Library/LaunchAgents/com.marlon.backup.plist
> launchctl bootstrap gui/$(id -u)
~/Library/LaunchAgents/com.marlon.backup.plist # cargar
> launchctl kickstart -k gui/$(id -u)/com.marlon.backup # ejecutar ya,
para probar
> launchctl print gui/$(id -u)/com.marlon.backup # estado y
último código de salida
> launchctl bootout gui/$(id -u)/com.marlon.backup # descargar
> launchctl list | grep -v com.apple # agentes de
terceros cargados
```

Clave	Para qué sirve
StartInterval	Cada N segundos (ej. 3600 = cada hora)
StartCalendarInterval	A una hora concreta; si el Mac dormía, se ejecuta al despertar
RunAtLoad	Al iniciar sesión
WatchPaths	Cuando cambia un archivo o carpeta
KeepAlive	Relanzar si el proceso muere (servicios)

cron: sigue funcionando

`cron` sigue disponible y es más rápido de configurar, aunque no recupera ejecuciones perdidas si el Mac estaba dormido. Puede necesitar Acceso total al disco para `/usr/sbin/cron`.

```
> crontab -e
# min hora día mes díasemana comando
30 21 * * 1-5 /Users/marlon/bin/backup_proyectos.zsh
*/15 * * * * /Users/marlon/bin/comprobar_servidor.zsh
> crontab -l
```

Vigilar carpetas

```
> brew install fswatch
> fswatch -o ~/Downloads | xargs -n1 -I{} ~/bin/ordenar_descargas.zsh
```

Tu entorno ideal

Homebrew, un .zshrc cuidado y las herramientas modernas que hacen la consola agradable.

Homebrew

Homebrew es el gestor de paquetes de facto en macOS. Se instala con el comando oficial publicado en `brew.sh` y, en Mac con Apple Silicon, vive en `/opt/homebrew`.

```
> brew install git jq ripgrep fd bat eza fzf zoxide btop tldr
> brew install --cask iterm2 visual-studio-code rectangle # apps
gráficas
> brew update && brew upgrade && brew cleanup # mantenimiento
> brew outdated ; brew leaves #
desactualizados / instalados a mano
> brew info postgresql@16 ; brew deps --tree eza
> brew services start postgresql@16 # servicios con
launchd
> brew doctor # diagnóstico
```

Brewfile: tu Mac reproducible

```
> brew bundle dump --file=~/.Brewfile # exporta todo lo instalado
> brew bundle --file=~/.Brewfile # en un Mac nuevo: lo instala
todo
```

Herramientas modernas recomendadas

Clásico	Moderno	Ventaja
<code>ls</code>	<code>eza</code>	Colores, iconos, vista árbol y estado git
<code>cat</code>	<code>bat</code>	Resaltado de sintaxis y números de línea
<code>cd</code>	<code>zoxide (z)</code>	Salta a carpetas frecuentes con parte del nombre
<code>Ctrl+R</code>	<code>fzf</code>	Búsqueda difusa en historial y archivos
<code>top</code>	<code>htop</code>	Panel visual de CPU, memoria, red y disco
<code>man</code>	<code>tlldr</code>	Ejemplos prácticos en lugar de referencia exhaustiva
<code>find / grep</code>	<code>fd / rg</code>	Velocidad y sintaxis sencilla

Un ~/.zshrc profesional

```
# ----- Historial -----
HISTFILE=~/.zsh_history; HISTSIZE=100000; SAVEHIST=100000
setopt SHARE_HISTORY HIST_IGNORE_ALL_DUPS HIST_IGNORE_SPACE
EXTENDED_HISTORY

# ----- Comportamiento -----
setopt AUTO_CD AUTO_PUSHD PUSHD_IGNORE_DUPS EXTENDED_GLOB
INTERACTIVE_COMMENTS
autoload -Uz compinit && compinit
zstyle ':completion:*' matcher-list 'm:{a-z}={A-Za-z}' # autocompletar
sin mayúsculas
zstyle ':completion:*' menu select
autoload -U zmv edit-command-line
zle -N edit-command-line; bindkey '^X^E' edit-command-line
export EDITOR="code -w"

# ----- PATH -----
eval "$(opt/homebrew/bin/brew shellenv)"
path=(~/bin $path)

# ----- Alias -----
alias ll='eza -la --git --group-directories-first'
alias cat='bat --paging=never'
alias ip='ipconfig getifaddr en0'
alias ippub='curl -s https://ifconfig.me; echo'
alias flushdns='sudo dscacheutil -flushcache; sudo killall -HUP
mDNSResponder'
alias ocultos='defaults write com.apple.finder AppleShowAllFiles -bool
true; killall Finder'
alias puertos='lsof -nP -iTCP -sTCP:LISTEN'
alias update='brew update && brew upgrade && brew cleanup; softwareupdate
-l'

# ----- Funciones -----
mkcd() { mkdir -p "$1" && cd "$1"; }
cdf() { cd "$(osascript -e 'tell app "Finder" to POSIX path of
(insertion location as alias)')'; }
extraer(){ case $1 in *.tar.gz|*.tgz) tar xzf $1;; *.zip) ditto -x -k $1
.;; *.dmg) hdiutil attach $1;; esac }
puerto() { lsof -nP -i :$1; }
avisa() { "$@"; local r=$?; osascript -e "display notification \"$1
terminó ($r)\" with title \"Terminal\""; return $r; }

# ----- Herramientas -----
```

```
eval "$(zoxide init zsh)"
source <(fzf --zsh)
```

```
> source ~/.zshrc      # aplica cambios sin abrir otra ventana
```

FUNCIÓN ESTRELLA: CDF

`cdf` te lleva en la Terminal a la carpeta que tienes abierta en la ventana delantera del Finder. Y su pareja natural es `open .`, que hace justo lo contrario. Entre las dos, ratón y teclado dejan de ser mundos separados.

Un prompt informativo

Para un prompt con rama de git, tiempo de ejecución y estado del último comando, las opciones más populares son **Starship** (`brew install starship` y añadir `eval "$(starship init zsh)"`) o **Powerlevel10k**. Si prefieres no instalar nada, zsh trae su propio sistema:

```
autoload -Uz vcs_info; precmd() { vcs_info }
zstyle ':vcs_info:git:*' formats ' %F{yellow}(%b)%f'
setopt PROMPT_SUBST
PROMPT='%F{green}%~%f${vcs_info_msg_0_} %(?.%F{white}.%F{red})>%f '
```

Cierre

La Terminal premia la constancia. No intentes memorizar este libro: elige tres comandos por semana, úsalos hasta que salgan solos y añade lo que repitas a tu `.zshrc` como alias o función. En pocos meses tendrás un entorno hecho a tu medida y una velocidad de trabajo que ninguna interfaz gráfica puede igualar.

APÉNDICE A

Chuleta final

Los comandos esenciales de este libro, agrupados para consulta rápida.

zsh

<code>sudo !!</code>	repetir con sudo
<code>!\$</code>	último argumento
<code>^a^b</code>	repetir sustituyendo
<code>**/*(.Lm+100)</code>	> 100 MB recursivo
<code>*(.om[1,5])</code>	5 más recientes
<code>\${f:t} \${f:r} \${f:e}</code>	nombre, raíz, ext.
<code>zmvm -n '(*).a' '\$1.b'</code>	renombrar masivo
<code>cd - / dirs -v</code>	carpetas previas

Tuberías

<code>&> f</code>	stdout+stderr
<code> &</code>	tubería con errores
<code> sudo tee -a f</code>	escribir con sudo
<code>diff <(a) <(b)</code>	sustitución procesos
<code>xargs -0 -P4</code>	paralelo y seguro

Buscar

<code>mdfind -name x</code>	Spotlight por nombre
<code>mdfind -onlyin ~ "q"</code>	en una carpeta
<code>mdls -name</code>	origen
<code>KMDItemWhereFroms</code>	descarga
<code>find . -mtime -1</code>	últimas 24 h
<code>rg -t py "x" / fd -e pdf</code>	modernos

Texto

<code>sed -i '' 's/a/b/g'</code>	editar en sitio
<code>awk -F';' '{s+=\$4}END{print s}'</code>	sumar
<code>sort uniq -c sort -rn</code>	ranking
<code>column -t -s';'</code>	alinear CSV
<code>jq -r '.[[]] @csv'</code>	JSON → CSV
<code>plutil -p f.plist</code>	leer plist

macOS

<code>open -R f</code>	mostrar en Finder
<code>pbpaste ... pbcopy</code>	portapapeles
<code>caffeinate -i cmd</code>	no dormir
<code>screencapture -ic</code>	captura a portap.
<code>sips -Z 1200 *.jpg</code>	redimensionar
<code>textutil -convert docx</code>	convertir docs
<code>say "texto"</code>	voz
<code>pmset -g assertions</code>	qué impide reposo

defaults

<code>defaults read dom clave</code>	leer
<code>defaults write dom k -bool true</code>	escribir
<code>defaults export dom f.plist</code>	respaldar
<code>killall Finder / Dock</code>	aplicar

Procesos

<code>lsof -i :8069</code>	quién usa el puerto
<code>lsof -nP -iTCP -sTCP:LISTEN</code>	puertos abiertos
<code>top -o cpu</code>	por CPU
<code>pkill -f patrón</code>	matar por patrón
<code>log show --last 10m</code>	registro
<code>sudo powermetrics</code>	consumo

Discos

<code>du -hd1 sort -hr</code>	qué ocupa
<code>diskutil list</code>	discos
<code>cp -c a b</code>	clon APFS
<code>ditto -c -k --keepParent</code>	zip nativo
<code>rsync -avhn --delete</code>	espejo (simulado)
<code>xattr -d</code>	quitar
<code>com.apple.quarantine</code>	cuarentena
<code>tmutil startbackup</code>	Time Machine

Red

<code>ipconfig getifaddr en0</code>	IP local
<code>networkQuality</code>	test velocidad
<code>dig +short dominio</code>	DNS
<code>nc -zv host 443</code>	puerto abierto
<code>ssh -L 8069:localhost:8069</code>	túnel
<code>networksetup -setdnsservers</code>	cambiar DNS

Seguridad

<code>security find-generic-password -w secreto</code>	
<code>spctl --assess -vv App.app</code>	Gatekeeper
<code>codesign -dv App.app</code>	firma
<code>fdesetup status</code>	FileVault
<code>openssl rand -base64 24</code>	contraseña

Automatización

<code>osascript -e 'display notification'</code>	aviso
<code>shortcuts run "X"</code>	Atajos
<code>launchctl bootstrap gui/\${id -u}</code>	cargar agente
<code>launchctl kickstart -k</code>	ejecutar ya
<code>crontab -e</code>	tareas cron
<code>brew bundle dump</code>	Brewfile

Debajo del Finder hay un Unix completo.

Este libro te da las llaves.

Si usas un Mac a diario, ya sabes lo que se puede hacer con el ratón. Este libro te enseña todo lo que no: buscar en Spotlight con precisión quirúrgica, convertir cientos de imágenes en segundos, descubrir qué proceso bloquea un puerto, cambiar ajustes que Apple no muestra y programar tareas que se ejecutan solas mientras duermes.

Escrito para profesionales que ya conocen lo básico y quieren dominar la Terminal de macOS con ejemplos reales y listos para copiar.

- › zsh avanzado: globbing con calificadores, modificadores y zmv
- › Los comandos exclusivos de Apple: mdfind, defaults, sips, pbcopy, caffeinate
- › Diagnóstico de procesos, memoria, energía y el registro unificado
- › Discos, imágenes, clones APFS, rsync y Time Machine
- › Redes, curl, SSH, llavero, firmas y Touch ID para sudo
- › Automatización con scripts, AppleScript, Atajos y launchd

```
› echo "¿Listo?" | say && open -a Terminal
```

Marlon Falcón Hernández

Especialista en ERP y automatización · Valencia



LA CONSOLA DEL MAC · 2026