
RA7. Llenguatges de guions

Índex

0. Bash vs. PowerShell: dues filosofies	1
1. Estructures del llenguatge	1
1.1. Variables i tipus de dades	1
1.2. Estructures de dades: arrays i diccionaris	2
1.3. Estructures de control	2
1.4. Funcions i pas de paràmetres	3
2. Creació i depuració d'scripts	4
2.1. Creació: capçalera i permisos (Bash)	4
2.2. Creació en PowerShell: política d'execució	4
2.3. Depuració d'errors sintàctics	5
2.4. Depuració d'errors d'execució (trampes i codis de sortida)	5
3. Interpretació d'scripts del sistema. Aplicacions	6
3.1. GNU/Linux: unitats systemd i scripts d'arrencada	6
3.2. Windows: perfils i tasques programades	6
4. Utilització d'extensions d'ordres per a tasques d'administració	7
4.1. Mòduls PowerShell	7
4.2. Llibreries de funcions pròpies en Bash	7
5. Scripts per a l'administració de comptes, processos i serveis	8
5.1. Comptes d'usuari	8
5.2. Processos	8
5.3. Serveis del sistema operatiu	9
5.4. Automatització de tasques	9
5.5. Documentació dels scripts	10
6. Implantació de guions en sistemes lliures i propietaris	10
Recursos	11

Aquest document treballa l'scripting en dos mons: el **lliure** (Bash, sobre GNU/Linux) i el **propietari** (PowerShell, sobre Windows). L'objectiu no és només aprendre sintaxi, sinó saber quan i com un guió és l'eina adequada per administrar un sistema: comptes d'usuari, processos, serveis i tasques automatitzades.

0. Bash vs. PowerShell: dues filosofies

	Bash (lliure)	PowerShell (propietari*)
Model de dades	Text pla (streams)	Objectes .NET
Ordres	Binaris externs (ls, grep...)	Cmdlets Verb-Noun
Extensió de fitxer	.sh	.ps1
Shebang	#!/bin/bash	No aplica (s'invoca amb l'interpret)
Multiplataforma	Nativament Unix/Linux	PowerShell 7+ és multiplataforma

* Des de PowerShell 7, Microsoft n'ha alliberat el codi (MIT license) i funciona sobre Linux i macOS, però el manual el tracta com a representant del món "propietari" perquè és l'eina d'administració per defecte de Windows Server i AD.

NOTA

El fet que Bash treballi amb text i PowerShell amb objectes és la diferència conceptual més important. A Bash, `ps aux | grep bash` filtra línies de text. A PowerShell, `Get-Process | Where-Object {$_.Name -eq "pwsh"}` filtra objectes Process reals, amb propietats tipades (.Id, .CPU, .Path...).

1. Estructures del llenguatge

1.1. Variables i tipus de dades

Bash --- tot és text per defecte, sense declaració de tipus:

```
nom="Ramon"
edat=42
readonly VERSIO="1.0"           # constant
declare -i comptador=0          # força el tipus enter
export PATH_LOG="/var/log/app.log" # variable d'entorn
```

PowerShell --- tipat dinàmic amb objectes .NET reals:

```
$nom = "Ramon"
$edat = 42
[int]$comptador = 0
$PATH_LOG = "/var/log/app.log"
Set-Variable -Name VERSIO -Value "1.0" -Option ReadOnly
```

1.2. Estructures de dades: arrays i diccionaris

```
# Bash
serveis=("ssh" "nginx" "mariadb")
echo "${serveis[1]}"           # nginx
echo "${serveis[@]}"           # tots els elements

declare -A usuaris=( [ramon]="professor" [alex]="alumne" )
echo "${usuaris[ramon]}"       # professor
```

```
# PowerShell
$serveis = @("ssh", "nginx", "mariadb")
$serveis[1]           # nginx

$usuaris = @{ ramon = "professor"; alex = "alumne" }
$usuaris["ramon"]     # professor
```

1.3. Estructures de control

```
# Condicional
if [ "$edat" -ge 18 ]; then
    echo "major d'edat"
elif [ "$edat" -eq 0 ]; then
    echo "no vàlid"
else
    echo "menor d'edat"
fi

# Selecció múltiple
case "$1" in
    start) echo "Arrencant...";;
    stop)  echo "Aturant...";;
    *)     echo "Ús: $0 {start|stop}";;
esac

# Bucles
for s in "${serveis[@]}"; do echo "$s"; done
i=0
while [ $i -lt 5 ]; do echo $i; ((i++)); done
```

```
# Condicional
if ($edat -ge 18) {
    "major d'edat"
} elseif ($edat -eq 0) {
    "no vàlid"
} else {
    "menor d'edat"
}
```

```
# Selecció múltiple
switch ($args[0]) {
    "start" { "Arrencant..." }
    "stop" { "Aturant..." }
    default { "Ús: $($MyInvocation.MyCommand.Name) {start|stop}" }
}

# Bucles
foreach ($s in $serveis) { $s }
for ($i = 0; $i -lt 5; $i++) { $i }
```

1.4. Funcions i pas de paràmetres

```
reinicia_servei() {
    local servei="$1"
    if systemctl is-active --quiet "$servei"; then
        systemctl restart "$servei"
        echo "Reiniciat: $servei"
    else
        echo "Error: $servei no actiu" >&2
        return 1
    fi
}
reinicia_servei "nginx"
```

```
function Reinicia-Servei {
    param(
        [Parameter(Mandatory)][string]$Servei
    )
    if ((Get-Service -Name $Servei).Status -eq "Running") {
        Restart-Service -Name $Servei
        Write-Output "Reiniciat: $Servei"
    } else {
        Write-Error "$Servei no actiu"
    }
}
Reinicia-Servei -Servei "Spooler"
```

2. Creació i depuració d'scripts

2.1. Creació: capçalera i permisos (Bash)

```
#!/bin/bash
#
# Nom:      backup_logs.sh
# Autor:    Ramon López
# Data:     2026-06-20
# Objectiu: Comprimeix i arxiva els logs de /var/log/app
# Ús:       ./backup_logs.sh [directori_desti]

set -euo pipefail # atura el script davant errors, variables no
↳ definides o pipes fallits
```

```
chmod +x backup_logs.sh
./backup_logs.sh /mnt/backups
```

`set -euo pipefail` és probablement la línia més important de qualsevol script Bash professional:

- `-e`: surt si qualsevol ordre retorna un error.
- `-u`: error si s'usa una variable no definida.
- `-o pipefail`: un pipe (`cmd1 | cmd2`) falla si **qualsevol** de les dues ordres falla, no només l'última.

2.2. Creació en PowerShell: política d'execució

```
# Comentari de capçalera basat en comment-based help (veure secció
↳ 5.4)
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
.\Backup-Logs.ps1 -Desti "D:\Backups"
```

Per defecte, Windows bloqueja l'execució d'scripts (Restricted). Cal establir una política com `RemoteSigned` (permet scripts locals, exigeix signatura als descarregats d'internet) --- això mateix es tracta com a criteri d'implantació al bloc 6.

2.3. Depuració d'errors sintàctics

Eina	Ús
<code>bash -n script.sh</code>	Comprova la sintaxi sense executar
<code>bash -x script.sh</code>	Traça d'execució (mostra cada ordre abans d'executar-la)
<code>shellcheck script.sh</code>	Analitzador estàtic extern, detecta errors habituals i mals hàbits
<code>PSScriptAnalyzer</code>	Equivalent a ShellCheck per a PowerShell
<code>Set-PSDebug -Trace 1</code>	Traça d'execució en PowerShell

```
bash -x backup_logs.sh
```

```
+ set -euo pipefail
+ DEST=/mnt/backups
+ tar czf /mnt/backups/app-2026-09-06.tar.gz /var/log/app
```

```
Set-PSDebug -Trace 1
.\Backup-Logs.ps1
Set-PSDebug -Off
```

2.4. Depuració d'errors d'execució (trampes i codis de sortida)

```
trap 'echo "Error a la línia $LINENO" >&2' ERR

ordre_que_pot_fallar
echo "Codi de sortida: $?"
```

```
try {
    Get-Service -Name "ServeiInexistent" -ErrorAction Stop
}
catch {
    Write-Error "Error capturat: $($_.Exception.Message)"
}
finally {
    Write-Output "Bloc finally sempre executat"
}
```

3. Interpretació d'scripts del sistema. Aplicacions

Un tècnic de sistemes passa més temps **llegint** scripts d'altri que escrivint-ne de nous. Cal saber identificar patrons habituals als scripts del mateix sistema operatiu.

3.1. GNU/Linux: unitats systemd i scripts d'arrencada

```
# /etc/systemd/system/monitor-disc.service
[Unit]
Description=Monitor d'espai en disc
After=network.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/monitor-disc.sh
User=root

[Install]
WantedBy=multi-user.target
```

Interpretació: ExecStart invoca un script extern; Type=oneshot indica que s'executa i acaba (típic de scripts, no de dimonis persistents); WantedBy determina en quin *target* d'arrencada s'habilita.

També cal saber llegir:

- /etc/profile.d/*.sh: scripts que carreguen l'entorn de tots els usuaris a l'inici de sessió.
- /etc/cron.d/, /etc/cron.daily/: scripts programats pel sistema.
- Scripts d'initramfs o /etc/init.d/ en sistemes amb SysVinit encara vigent.

3.2. Windows: perfils i tasques programades

```
# $PROFILE --- s'executa a l'inici de cada sessió PowerShell
notepad $PROFILE
```

```
# Consultar una tasca programada i el script que invoca
Get-ScheduledTask -TaskName "ActualitzaInventari" |
    Select-Object TaskName, State
(Get-ScheduledTask -TaskName "ActualitzaInventari").Actions
```

NOTA

Interpretar un script no és només llegir-ne el codi: cal identificar **qui l'invoca** (systemd, cron, Task Scheduler, un altre script), **amb quins privilegis** s'executa i **quines dependències** (altres binaris, variables d'entorn, fitxers de configuració) necessita per funcionar correctament.

4. Utilització d'extensions d'ordres per a tasques d'administració

4.1. Mòduls PowerShell

```
Get-Module -ListAvailable ActiveDirectory
Import-Module ActiveDirectory
Get-ADUser -Filter {Enabled -eq $false} | Select-Object Name,
↪ LastLogonDate
```

Els mòduls (Import-Module) són l'equivalent PowerShell de les llibreries: paquets de cmdlets especialitzats (AD, DNS, DHCP, Hyper-V...) que estenen l'interpret sense escriure'ls des de zero.

4.2. Llibreries de funcions pròpies en Bash

```
# /usr/local/lib/funcions_admin.sh
comprova_root() {
    [ "$EUID" -eq 0 ] || { echo "Cal executar com a root" >&2; exit 1; }
}
log() {
    echo "$(date '+%F %T') - $1" >> /var/log/scripts_admin.log
}
```

```
#!/bin/bash
source /usr/local/lib/funcions_admin.sh # o bé: .
↪ /usr/local/lib/funcions_admin.sh
comprova_root
log "Script iniciat"
```

source (o el seu àlies `.`) carrega les funcions al context actual, sense obrir un subprocés nou: és el mecanisme bàsic per consultar i reutilitzar llibreries de funcions en Bash.

5. Scripts per a l'administració de comptes, processos i serveis

5.1. Comptes d'usuari

```
#!/bin/bash
# alta_usuari.sh --- dona d'alta un usuari i el crea al grup indicat
set -euo pipefail
usuari="$1"; grup="${2:-alumnes}"

useradd -m -s /bin/bash -G "$grup" "$usuari"
passwd -e "$usuari"          # força el canvi de contrasenya al primer
    ↪ login
echo "Usuari $usuari creat i afegit a $grup"
```

```
# Alta-Usuari.ps1
param(
    [Parameter(Mandatory)][string]$Usuari,
    [string]$Grup = "Alumnes"
)
$pwd = ConvertTo-SecureString "CanviaAquestaContrasenya!"
    ↪ -AsPlainText -Force
New-LocalUser -Name $Usuari -Password $pwd
    ↪ -PasswordNeverExpires:$false
Add-LocalGroupMember -Group $Grup -Member $Usuari
Write-Output "Usuari $Usuari creat i afegit a $Grup"
```

5.2. Processos

```
ps aux --sort=-%cpu | head -5          # 5 processos que més CPU
    ↪ consumeixen
pgrep -f "python3 servidor.py"        # localitzar per nom
kill -SIGTERM "$(pgrep -f servidor.py)"
```

```
Get-Process | Sort-Object CPU -Descending | Select-Object -First 5
Stop-Process -Name "notepad" -Force
```

5.3. Serveis del sistema operatiu

```
#!/bin/bash
# gestiona_servei.sh start|stop|status <nom_servei>
accio="$1"; servei="$2"
case "$accio" in
    start) systemctl start "$servei"  && echo "$servei iniciat";;
    stop)  systemctl stop  "$servei"  && echo "$servei aturat";;
    status) systemctl status "$servei" --no-pager;;
    *) echo "Ús: $0 {start|stop|status} <servei>"; exit 1;;
esac
```

```
# Gestiona-Servei.ps1
param(
    [ValidateSet("Start","Stop","Status")][string]$Accio,
    [Parameter(Mandatory)][string]$Servei
)
switch ($Accio) {
    "Start" { Start-Service -Name $Servei; "$Servei iniciat" }
    "Stop"  { Stop-Service  -Name $Servei; "$Servei aturat" }
    "Status" { Get-Service   -Name $Servei }
}
```

5.4. Automatització de tasques

```
# crontab -e
# Cada dia a les 2:30, executa la còpia de seguretat
30 2 * * * /usr/local/bin/backup_logs.sh /mnt/backups >>
↳ /var/log/backup.log 2>&1
```

```
# Alternativa moderna: systemd timer
# /etc/systemd/system/backup.timer
[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true

[Install]
WantedBy=timers.target
```

```
# Programar la mateixa tasca amb Task Scheduler des de PowerShell
$accio = New-ScheduledTaskAction -Execute "pwsh.exe" -Argument
↳ "-File C:\Scripts\Backup-Logs.ps1"
$disparador = New-ScheduledTaskTrigger -Daily -At 2:30am
Register-ScheduledTask -TaskName "BackupLogs" -Action $accio -Trigger
↳ $disparador
```

5.5. Documentació dels scripts

Bash: capçalera de comentaris (autor, data, objectiu, ús) com a l'apartat 2.1, més comentaris explicatius abans de blocs no evidents.

PowerShell té un sistema natiu de documentació integrat, el *comment-based help*:

```
<#
.SYNOPSIS
    Gestiona l'estat d'un servei de Windows.
.DESCRIPTION
    Inicia, atura o consulta l'estat d'un servei indicat pel nom.
.PARAMETER Servei
    Nom del servei de Windows a gestionar.
.EXAMPLE
    .\Gestiona-Servei.ps1 -Accio Start -Servei Spooler
#>
param(...)
```

Amb aquesta capçalera, `Get-Help .\Gestiona-Servei.ps1 -Full` genera automàticament l'ajuda de l'script, exactament igual que amb un cmdlet natiu.

6. Implantació de guions en sistemes lliures i propietaris

Aspecte	Linux / lliure	Windows / propietari
Permisos d'execució	<code>chmod +x script.sh</code>	No cal per si mateix; depèn de la política d'execució
Control de qui pot executar	Permisos POSIX (usuari/grup/altres) + <code>sudo</code>	<code>Set-ExecutionPolicy</code> , signatura digital de codi
Ubicació estàndard	<code>/usr/local/bin</code> , <code>/usr/local/sbin</code>	Carpeta pròpia + variable <code>PATH</code> , o mòdul instal·lat a <code>\$env:PSModulePath</code>
Distribució massiva	Paquet <code>.deb/.rpm</code> , Ansible, <code>scp</code> + <code>cron</code>	GPO, PowerShell DSC, Intune, signatura amb <code>Set-AuthenticodeSignature</code>
Comprovar integritat	<code>sha256sum</code>	<code>Get-FileHash</code>

```
# Signar un script per poder-lo distribuir amb política AllSigned
$cert = Get-ChildItem Cert:\CurrentUser\My -CodeSigningCert
Set-AuthenticodeSignature -FilePath .\Backup-Logs.ps1 -Certificate
↪ $cert
```

NOTA

“Implantar” un guió no acaba en copiar-lo al servidor: inclou triar **on** viu (repositori, control de versions), **com** s'invoca (manualment, per cron/Task Scheduler, com a servei), **amb quins privilegis** i **com es revisa** si deixa de funcionar (logs, alertes). Aquest és exactament el punt on el scripting es converteix en administració de sistemes real.

Recursos

- `man bash / help` (Bash intern) --- referència completa del llenguatge.
- `Get-Help about_*` en PowerShell --- p. ex. `Get-Help about_Functions`.
- [ShellCheck](#) --- analitzador estàtic per a Bash.
- Mòdul PSScriptAnalyzer --- analitzador estàtic per a PowerShell.
- `man systemd.service`, `man systemd.timer` --- unitats de systemd.
- [Shell scripts](#)

Versions d'aquest document

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

[Domini Públic \(CC0\)](#)