MÉMO TECHNIQUE

Automatisation & scripting multiplateforme

Scripts PowerShell

Cmdlets · Pipeline d'objets · Fonctions · DevOps

Objet	Fondamentaux du scripting PowerShell et usages DevOps
Public	Développeurs, DevOps, profils techniques
Auteur	OSD — M. Slimani
Version	v1.0 — 17/06/2026

Situer PowerShell

PowerShell est à la fois un *shell* (interpréteur de commandes) et un **langage de script**. Sa particularité : le pipeline ne fait pas circuler du texte brut comme Bash, mais des **objets .NET** typés, avec leurs propriétés et leurs méthodes.

Depuis **PowerShell 7** (commande `pwsh`), il est **multiplateforme** : Windows, Linux et macOS. Il complète la connaissance de Bash et s'intègre naturellement aux chaînes CI/CD (build, déploiement, administration de serveurs Windows).

> ℹ️ **Lien avec la formation Git & CI/CD**
> Comme tout code, les scripts `*.ps1` se versionnent dans Git. Ils servent souvent de scripts d'étape dans un pipeline (build, packaging, déploiement) au même titre qu'un script Bash.

1. Les fondamentaux

Cmdlets : la convention Verbe-Nom

Les commandes PowerShell (les *cmdlets*) suivent toutes la forme **Verbe-Nom** : le verbe décrit l'action, le nom décrit la cible. Cette régularité rend les commandes prévisibles.

```
PS> Get-Process          # lister les processus
PS> Get-Service          # lister les services
PS> Set-Location C:\app  # changer de répertoire (alias: cd)
PS> Get-Command -Verb Get  # toutes les cmdlets dont le verbe est 'Get'
```

> ✓ **Le réflexe à acquérir**
> Trois cmdlets suffisent pour explorer : `Get-Command` (que puis-je faire ?), `Get-Help` (comment l'utiliser ?) et `Get-Member` (quelles propriétés/méthodes possède cet objet ?).

Variables et types

Une variable est préfixée par `$`. Le typage est dynamique mais peut être contraint explicitement.

```
$nom = "OSD"
$port = 8080
[int]$compteur = 0          # type contraint
$serveurs = @("web1", "web2", "db1")   # tableau
$config = @{ Host = "localhost"; Port = 5432 }  # table de hachage
```

Le pipeline d'objets

Le caractère `|` transmet l'objet produit par une cmdlet à la suivante. Comme ce sont des objets et non du texte, on filtre et on sélectionne par **propriété**, sans parsing fragile.

```
PS> Get-Process | Where-Object { $_.CPU -gt 100 } | Select-Object Name, CPU

# $_ représente l'objet courant qui traverse le pipeline
```

> ⚠️ **Différence clé avec Bash**
>
> En Bash, un pipe transporte du **texte** que l'on découpe avec grep/awk/sed. En PowerShell, il transporte des **objets structurés** : on accède directement à `$_.Name` ou `$_.Length` sans extraction manuelle.

2. Structures de contrôle

La syntaxe des conditions et des boucles est proche du C/Java, avec des accolades.

Conditions

```
if ($port -eq 8080) {
    Write-Host "Port par défaut"
} elseif ($port -gt 1024) {
    Write-Host "Port utilisateur"
} else {
    Write-Host "Port système"
}
```

> ℹ️ **Opérateurs de comparaison**
>
> PowerShell n'utilise pas `==` ni `>` mais des opérateurs textuels : `-eq`, `-ne`, `-gt`, `-lt`, `-ge`, `-le`, `-like` (jokers) et `-match` (regex).

Boucles

```
foreach ($serveur in $serveurs) {
    Write-Host "Vérification de $serveur"
}

# Forme pipeline équivalente
$serveurs | ForEach-Object { Write-Host "Vérification de $_" }

for ($i = 0; $i -lt 5; $i++) { Write-Host $i }
while ($compteur -lt 3) { $compteur++ }
```

3. Fonctions et scripts

Une fonction encapsule une logique réutilisable. Le bloc `param()` déclare les paramètres typés et leurs valeurs par défaut — clé d'un code modulaire et réutilisable.

```
function Test-Connexion {
    param(
        [Parameter(Mandatory)] [string]$Hote,
        [int]$Port = 443
    )
    $resultat = Test-NetConnection -ComputerName $Hote -Port $Port
    return $resultat.TcpTestSucceeded
}

PS> Test-Connexion -Hote "osdigitalis.com" -Port 443
```

✓ **Bonne pratique : verbes approuvés**

Nommez vos fonctions avec un verbe approuvé (`Get-Verb` donne la liste). `Test-`, `Get-`, `New-`, `Set-` : vos fonctions s'intègrent ainsi à l'écosystème comme des cmdlets natives.

4. Cmdlets essentielles

Une poignée de cmdlets couvre l'essentiel du traitement de données dans le pipeline.

Cmdlet	Alias	Rôle
`Where-Object`	`?`	Filtrer les objets selon une condition
`Select-Object`	`select`	Choisir des propriétés / les N premiers
`ForEach-Object`	`%`	Exécuter une action sur chaque objet
`Sort-Object`	`sort`	Trier par propriété
`Measure-Object`	—	Compter, sommer, faire des moyennes
`Export-Csv`	—	Exporter le résultat vers un fichier CSV

```
# Top 3 des processus les plus gourmands, exporté en CSV
Get-Process |
    Sort-Object CPU -Descending |
    Select-Object -First 3 Name, CPU |
    Export-Csv -Path top3.csv -NoTypeInformation
```

5. PowerShell en DevOps

Au-delà de l'administration, PowerShell brille pour automatiser des tâches d'intégration : appels d'API REST, manipulation de JSON, gestion de fichiers et orchestration d'étapes de pipeline.

```
# Appel d'une API REST et lecture directe du JSON (devenu objet)
```

```powershell
$reponse = Invoke-RestMethod -Uri "https://api.github.com/repos/osd/osd-blog"
Write-Host "Étoiles : $($reponse.stargazers_count)"

# Conversions JSON <-> objet
$obj = Get-Content config.json | ConvertFrom-Json
$obj | ConvertTo-Json -Depth 5 | Set-Content config.json
```

Dans un pipeline CI/CD

GitHub Actions et GitLab CI savent exécuter des étapes en PowerShell (`shell: pwsh`). Un même script `*.ps1` peut ainsi tourner sur un runner Windows comme Linux, ce qui favorise la réutilisabilité.

6. Sécurité d'exécution

PowerShell encadre l'exécution des scripts par une **politique d'exécution** (*Execution Policy*) qui détermine quels scripts peuvent être lancés.

```powershell
PS> Get-ExecutionPolicy                # politique courante
PS> Set-ExecutionPolicy RemoteSigned   # scripts locaux OK, distants signés
```

Recommandation

Ne désactivez **jamais** la politique avec `Unrestricted` sur un poste de production. Préférez `RemoteSigned` : il exige que les scripts téléchargés soient signés numériquement, tout en laissant tourner vos scripts locaux. La signature de scripts s'appuie sur un certificat, dans la même logique que la signature des commits Git (GPG/SSH).

Récapitulatif

Notion	Élément clé	À retenir
Cmdlet	`Verbe-Nom`	Commandes régulières et prévisibles
Pipeline	`|`	Transporte des objets, pas du texte
Variable	`$nom`	Typage dynamique, contraignable
Objet courant	`$_`	Référence l'élément dans le pipeline
Fonction	`param()`	Code modulaire, paramètres typés
API REST	`Invoke-RestMethod`	JSON converti automatiquement en objet
Sécurité	`RemoteSigned`	Politique d'exécution recommandée

✓ La phrase à retenir

PowerShell fait circuler des **objets** (et non du texte) dans un pipeline, avec des cmdlets **Verbe-Nom** régulières. C'est cette orientation objet qui le distingue de Bash et le rend puissant pour l'automatisation DevOps multiplateforme.

ℹ️ Où Git intervient

Vos scripts `*.ps1` sont du code : versionnés dans un dépôt, revus en pull request et exécutés comme étapes de pipeline. La maîtrise de Git — objet de la formation — reste le socle commun.