MEMO TECHNIQUE

Les commandes essentielles, classées par catégorie

Scripting PowerShell

Aide · Fichiers · Pipeline · Contrôle · Erreurs · Processus · Réseau · Données

Objet	Commandes PowerShell par catégorie (commande, description, exemple)
Public	Développeurs, DevOps, profils techniques
Auteur	OSD — M. Slimani
Version	v1.0 — 17/06/2026

Situer PowerShell

PowerShell est à la fois un shell et un langage de script orienté objet. Là où un shell Unix manipule du texte, PowerShell fait circuler dans le pipeline des objets .NET typés : chaque commande renvoie des objets dont on peut filtrer, trier et exploiter directement les propriétés.

Toutes les commandes (les *cmdlets*) suivent la convention **Verbe-Nom** — `Get-`, `Set-`, `New-`, `Remove-` — ce qui rend l'ensemble prévisible et facile à découvrir. Multiplateforme depuis PowerShell 7 (Windows, Linux, macOS), il s'intègre naturellement dans les chaînes CI/CD et l'automatisation d'infrastructure.

> ℹ️ **La convention Verbe-Nom**
>
> Chaque cmdlet associe un verbe approuvé à un nom au singulier : `Get-Service`, `Stop-Process`. La liste des verbes valides s'obtient avec `Get-Verb`. Connaître le verbe suffit souvent à deviner la commande.

> ✓ **Le pipeline manipule des objets, pas du texte**
>
> C'est pourquoi `Where-Object` et `Select-Object` travaillent sur des propriétés (`CPU`, `Name`…) et non sur des colonnes de texte comme `awk`/`cut` sous Unix.

1. Aide & découverte

Le point de départ : trouver une commande, comprendre ce qu'elle fait et inspecter les objets qu'elle renvoie.

Commande	Description	Exemple
`Get-Help`	Affiche l'aide d'une commande	`Get-Help Get-Process -Examples`
`Get-Command`	Recherche les commandes disponibles	`Get-Command -Verb Get -Noun Service`
`Get-Member`	Liste propriétés et méthodes d'un objet	`Get-Process | Get-Member`
`Get-Alias`	Affiche les alias des cmdlets	`Get-Alias ls`
`Get-Verb`	Liste les verbes approuvés	`Get-Verb`
`Update-Help`	Met à jour l'aide locale	`Update-Help -Force`

2. Navigation & système de fichiers

Se déplacer dans l'arborescence et manipuler fichiers et dossiers.

Commande	Description	Exemple
`Get-ChildItem`	Liste le contenu d'un dossier (alias ls/dir)	`Get-ChildItem -Recurse -Filter *.log`
`Set-Location`	Change de répertoire courant (alias cd)	`Set-Location C:\Projets\osd`
`Get-Location`	Affiche le répertoire courant (alias pwd)	`Get-Location`
`Test-Path`	Teste l'existence d'un chemin	`Test-Path .\config.json`
`New-Item`	Crée un fichier ou un dossier	`New-Item -ItemType Directory logs`
`Copy-Item`	Copie fichier/dossier	`Copy-Item .\src -Destination .\bak -Recurse`
`Move-Item`	Déplace fichier/dossier	`Move-Item a.txt .\archive\`
`Remove-Item`	Supprime fichier/dossier	`Remove-Item .\tmp -Recurse -Force`
`Rename-Item`	Renomme un élément	`Rename-Item old.txt new.txt`

3. Contenu des fichiers

Lire, écrire et chercher dans le texte des fichiers.

Commande	Description	Exemple
`Get-Content`	Lit le contenu (alias cat/gc)	`Get-Content .\app.log -Tail 20`
`Set-Content`	Écrit en écrasant le fichier	`"v1.0" | Set-Content version.txt`
`Add-Content`	Ajoute à la fin du fichier	`Add-Content log.txt "nouvelle ligne"`
`Out-File`	Redirige une sortie vers un fichier	`Get-Process | Out-File procs.txt`
`Select-String`	Recherche un motif (équiv. grep)	`Select-String -Path *.log -Pattern ERROR`

4. Objets & pipeline

Le cœur de PowerShell : on enchaîne des cmdlets qui se passent des objets typés, pas du texte.

Commande	Description	Exemple
`Where-Object`	Filtre les objets sur une condition (alias ?)	`Get-Process | Where-Object CPU -gt 100`
`Select-Object`	Choisit des propriétés / un nombre de lignes	`Get-Process | Select-Object Name,CPU -First 5`
`Sort-Object`	Trie les objets	`Get-Process | Sort-Object CPU -Descending`
`ForEach-Object`	Action sur chaque objet (alias %)	`1..3 | ForEach-Object { $_ * 2 }`
`Measure-Object`	Compte, somme, moyenne…	`Get-Content f.txt | Measure-Object -Line`
`Group-Object`	Regroupe par propriété	`Get-Process | Group-Object Company`

> ✓ **Astuce — alias en interactif, noms complets en script**
> En ligne de commande, `?`, `%`, `ls` font gagner du temps. Dans un script versionné, préférez les noms complets (`Where-Object`, `ForEach-Object`) : c'est plus lisible et plus portable.

5. Variables, types & opérateurs

Stocker des valeurs, les typer et les comparer.

Commande	Description	Exemple
`$var = …`	Déclare/affecte une variable	`$count = 10`
`[type]`	Typage explicite (cast)	`[int]$n = "42"`
`-eq -ne -gt -lt`	Comparaison	`if ($n -gt 5) { "ok" }`
`-and -or -not`	Opérateurs logiques	`if ($a -and $b) { … }`
`-match`	Test par expression régulière	`"abc123" -match "\d+"`
`-like`	Test avec jokers (* ?)	`$nom -like "osd*"`
`@( ) / @{ }`	Tableau / table de hachage	`$h = @{ nom = "OSD" }`

6. Structures de contrôle

Conditions et boucles.

Commande	Description	Exemple
`if / elseif / else`	Branche conditionnelle	`if ($x) {…} elseif ($y) {…} else {…}`
`switch`	Aiguillage multi-valeurs	`switch ($v) { 1 {"un"} default {"?"} }`
`for`	Boucle à compteur	`for ($i=0; $i -lt 5; $i++) { $i }`
`foreach`	Itère une collection	`foreach ($f in $files) { $f.Name }`
`while / do`	Boucle conditionnelle	`while ($i -lt 3) { $i++ }`

7. Fonctions & paramètres

Factoriser et rendre le code réutilisable — la base d'un script propre.

Commande	Description	Exemple
`function`	Définit une fonction	`function Get-Sum { param($a,$b) $a+$b }`
`param( )`	Déclare les paramètres	`param([string]$Name, [int]$Age)`
`[CmdletBinding()]`	Active les fonctions avancées	`[CmdletBinding()] param()`
`[Parameter()]`	Contraint un paramètre	`[Parameter(Mandatory)] [string]$Path`

Commande	Description	Exemple
`return`	Renvoie une valeur	`return $result`

8. Gestion des erreurs

Capturer les exceptions et contrôler le comportement en cas d'échec.

Commande	Description	Exemple
`try / catch / finally`	Capture les exceptions	`try { … } catch { Write-Error $_ }`
`-ErrorAction`	Comportement local sur erreur	`Get-Item x -ErrorAction Stop`
`throw`	Lève une exception	`throw "Paramètre invalide"`
`$Error`	Pile des dernières erreurs	`$Error[0]`
`$ErrorActionPreference`	Politique globale d'erreur	`$ErrorActionPreference = "Stop"`

9. Processus & services

Piloter les processus et les services Windows — utile en exploitation.

Commande	Description	Exemple
`Get-Process`	Liste les processus (alias ps)	`Get-Process -Name node`
`Stop-Process`	Termine un processus	`Stop-Process -Name node -Force`
`Start-Process`	Lance un programme	`Start-Process notepad`
`Get-Service`	Liste les services	`Get-Service -Name W32Time`
`Restart-Service`	Redémarre un service	`Restart-Service -Name Spooler`

10. Réseau & web

Tester la connectivité et consommer des API — incontournable côté DevOps.

Commande	Description	Exemple
`Test-Connection`	Teste la connectivité (équiv. ping)	`Test-Connection google.com -Count 2`
`Test-NetConnection`	Teste un hôte et un port	`Test-NetConnection osdigitalis.com -Port 443`
`Invoke-WebRequest`	Requête HTTP brute (alias iwr)	`Invoke-WebRequest $url -OutFile f.json`
`Invoke-RestMethod`	Appel d'API REST (JSON automatique)	`Invoke-RestMethod api.github.com/users/osd`

11. Données & formats

Convertir, exporter et mettre en forme les données.

Commande	Description	Exemple
`ConvertTo-Json`	Sérialise un objet en JSON	`$obj | ConvertTo-Json -Depth 3`
`ConvertFrom-Json`	Désérialise du JSON en objet	`$txt | ConvertFrom-Json`
`Export-Csv`	Exporte des objets en CSV	`Get-Process | Export-Csv p.csv -NoTypeInformation`
`Import-Csv`	Lit un CSV en objets	`Import-Csv users.csv`
`Format-Table`	Affichage en tableau	`Get-Service | Format-Table -AutoSize`
`ConvertTo-Html`	Génère un rapport HTML	`Get-Process | ConvertTo-Html | Out-File r.html`

12. Modules & exécution

Étendre PowerShell et gérer l'environnement d'exécution.

Commande	Description	Exemple
`Get-Module`	Liste les modules	`Get-Module -ListAvailable`
`Import-Module`	Charge un module	`Import-Module Pester`
`Install-Module`	Installe depuis la galerie	`Install-Module Az -Scope CurrentUser`
`Get-ExecutionPolicy`	Affiche la politique de scripts	`Get-ExecutionPolicy -List`
`Set-ExecutionPolicy`	Définit la politique de scripts	`Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`
`$env:`	Variables d'environnement	`$env:PATH`

> **ExecutionPolicy n'est pas une barrière de sécurité**
> Elle évite l'exécution accidentelle de scripts, mais ne protège pas d'un acteur malveillant (contournable par `-ExecutionPolicy Bypass`). Pour un vrai contrôle : signez vos scripts (cf. module cybersécurité), restreignez les droits et auditez le code avant exécution.

Récapitulatif

Catégorie	À retenir	Commandes-clés
Aide & découverte	Trouver et comprendre une commande	Get-Help, Get-Command, Get-Member
Fichiers	Naviguer et manipuler l'arborescence	Get-ChildItem, Copy-Item, Test-Path
Contenu	Lire, écrire, chercher dans le texte	Get-Content, Set-Content, Select-String
Pipeline	Filtrer/trier des objets typés	Where-Object, Select-Object, Sort-Object
Contrôle	Conditions et boucles	if, switch, foreach, while
Fonctions	Code réutilisable et paramétré	function, param(), [CmdletBinding()]
Erreurs	Capturer et maîtriser les échecs	try/catch, -ErrorAction, throw
Processus/Services	Piloter l'exécution système	Get-Process, Get-Service, Restart-Service
Réseau/Web	Tester et consommer des API	Test-NetConnection, Invoke-RestMethod
Données	Convertir et exporter	ConvertTo-Json, Export-Csv, ConvertTo-Html
Modules/Exécution	Étendre et configurer	Import-Module, Install-Module, ExecutionPolicy

✓ La phrase à retenir

PowerShell = **cmdlets Verbe-Nom** + **pipeline d'objets**. On **découvre** avec `Get-Command/Get-Help/Get-Member`, on **filtre/trie** avec `Where/Sort/Select`, et on **automatise** avec des fonctions encadrées par `try/catch`.

ℹ️ Lien avec la formation

Les scripts PowerShell (build, déploiement, hooks) sont du code : ils se versionnent dans Git au même titre que l'applicatif, et alimentent les pipelines CI/CD. La signature de scripts est traitée dans le module cybersécurité.