Cours complet de PowerShell pour l'administration Windows

Formation en scripting et automatisation

28 septembre 2025

Table des matières

1 Introduction

Pourquoi PowerShell ?

PowerShell est le langage de scripting et la console d'administration de Microsoft. Il permet :

- d'automatiser les tâches d'administration système ;
- de gérer Active Directory, BitLocker, GPO, services, fichiers, etc. ;
- de travailler avec des objets (contrairement aux simples lignes de texte de CMD).

—

2 Bases de PowerShell

2.1 Variables et types de données

```
$nom = "Alice"
$age = 25
$estAdmin = $true
Write-Output "Utilisateur : $nom"
```

2.2 Opérateurs utiles

- Arithmétiques : `+ - * / %`
- Comparaison : `-eq, -ne, -lt, -gt, -like`
- Logiques : `-and, -or, -not`

2.3 Conditions

```
if ($age -ge 18) {
   Write-Output "Majeur"
} else {
   Write-Output "Mineur"
}
```

2.4 Boucles

```
$users = @("Alice","Bob","Claire")
foreach ($u in $users) {
   Write-Output "Utilisateur : $u"
}
```

2.5 Fonctions

```
function Afficher-Message($texte) {
   Write-Output "Message : $texte"
}
Afficher-Message "Bonjour le monde"
```

—

3 Gestion des fichiers et dossiers

3.1 Lister et créer des fichiers

```
# Lister fichiers d'un dossier
Get-ChildItem C:\Users

# Créer un fichier
"Bonjour" | Out-File "C:\Temp\test.txt"
```

3.2 Lire un fichier

```
$contenu = Get-Content "C:\Temp\test.txt"
Write-Output $contenu
```

---

4 Administration Windows avec PowerShell

4.1 Services

```
Get-Service
Restart-Service Spooler
```

4.2 Utilisateurs locaux

```
New-LocalUser "testuser" -Password (Read-Host -AsSecureString "Password")
```

4.3 Active Directory (AD)

```
# Nécessite module ActiveDirectory
Get-ADUser -Filter * | Select-Object Name, Enabled
```

---

5 BitLocker avec PowerShell

5.1 Vérifier l'état

```
Get-BitLockerVolume | Select-Object MountPoint, ProtectionStatus
```

5.2 Activer le chiffrement

```
Enable-BitLocker -MountPoint "C:" -EncryptionMethod XtsAes256 -UsedSpaceOnly `
  -TpmProtector
```

5.3 Récupération des clés

```
(Get-BitLockerVolume -MountPoint "C:").KeyProtector
```

---

6 TP pratiques

6.1 TP1 — Inventaire des utilisateurs locaux

— Écrire un script qui liste les utilisateurs locaux et exporte la liste dans un fichier texte.

6.2 TP2 — Vérification BitLocker

— Écrire un script qui affiche pour chaque volume : lettre de lecteur, état de protection et méthode de chiffrement.

6.3 TP3 — Supervision de services

— Écrire un script qui vérifie si le service `Spooler` est démarré, sinon le redémarre et écrit un log.

—

7 Gestion avancée d'Active Directory

7.1 Lister les utilisateurs AD

```
# Nécessite RSAT et module ActiveDirectory
Import-Module ActiveDirectory
Get-ADUser -Filter * | Select-Object Name, SamAccountName, Enabled
```

7.2 Créer un utilisateur AD

```
New-ADUser -Name "Jean Dupont" -SamAccountName jdupont ‘
 -UserPrincipalName jdupont@entreprise.local -Path "OU=Utilisateurs,DC=entreprise,DC=loc
 -AccountPassword (Read-Host -AsSecureString "Mot de passe") -Enabled $true
```

7.3 Ajouter un utilisateur à un groupe

```
Add-ADGroupMember -Identity "Admins" -Members jdupont
```

7.4 Lister les machines du domaine

```
Get-ADComputer -Filter * | Select-Object Name, Enabled
```

—

8 Gestion des GPO avec PowerShell

8.1 Lister les GPO existantes

```
Import-Module GroupPolicy
Get-GPO -All
```

8.2 Créer une nouvelle GPO

```
New-GPO -Name "GPO_Secu_BitLocker" -Comment "Active BitLocker sur postes du domaine"
```

8.3 Lier une GPO à une OU

```
New-GPLink -Name "GPO_Secu_BitLocker" -Target "OU=Postes,DC=entreprise,DC=local"
```

8.4 Sauvegarder et restaurer une GPO

```
# Sauvegarde
Backup-GPO -Name "GPO_Secu_BitLocker" -Path "C:\Backup\GPOs"

# Restauration
Restore-GPO -Name "GPO_Secu_BitLocker" -Path "C:\Backup\GPOs"
```

---

9 Logs et supervision avec PowerShell

9.1 Lire le journal des événements

```
Get-EventLog -LogName System -Newest 10
```

9.2 Filtrer les événements critiques

```
Get-EventLog -LogName Application -EntryType Error -Newest 20
```

9.3 Supervision continue

```
# Surveiller les erreurs système toutes les 10 secondes
while ($true) {
   Get-EventLog -LogName System -EntryType Error -Newest 1
   Start-Sleep -Seconds 10
}
```

9.4 Exporter les logs en CSV

```
Get-EventLog -LogName Security -Newest 50 | Export-Csv "C:\Logs\Security.csv" -NoTypeInf
```

---

10 Sécurité avancée avec PowerShell

10.1 BitLocker — rappel rapide

```
Get-BitLockerVolume | Select-Object MountPoint, ProtectionStatus, EncryptionMethod
```

10.2 Permissions NTFS

```
# Vérifier les permissions d'un dossier
Get-Acl "C:\Data"

# Modifier les permissions
$acl = Get-Acl "C:\Data"
```

```
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule("Utilisateurs","Re
$acl.SetAccessRule($rule)
Set-Acl "C:\Data" $acl
```

10.3 Auditer les comptes administrateurs

```
Get-LocalGroupMember -Group "Administrateurs"
```

—

11 TP pratiques avancés

11.1 TP4 — Création d'un utilisateur AD avec GPO BitLocker

- Créez un utilisateur AD nommé `testgpo`.
- Ajoutez-le à un groupe `Secu-BitLocker`.
- Créez une GPO qui applique BitLocker uniquement aux machines de ce groupe.

11.2 TP5 — Supervision de logs Windows

- Écrire un script qui enregistre toutes les erreurs du journal `System` dans un fichier `C:_erreurs.txt`.
- Le script doit s'exécuter en boucle toutes les 30 secondes.

11.3 TP6 — Audit sécurité

- Écrire un script qui liste :
  - Les utilisateurs AD désactivés.
  - Les membres du groupe Administrateurs locaux de chaque poste.
  - Les 20 dernières erreurs du journal Sécurité.
- Exporter les résultats en CSV pour reporting.
- —

12 Conclusion

Avec ce module enrichi, vous maîtrisez :
- La gestion Active Directory via PowerShell.
- La manipulation et l'automatisation des GPO.
- L'exploitation des journaux Windows pour supervision et audit.
- Des scripts de sécurité pour BitLocker et permissions NTFS.

Ces connaissances vous permettent d'être autonome pour écrire des scripts professionnels de gestion et de sécurité Windows.

—

13 Rappels essentiels

- Variables, conditions, boucles.
- Gestion des fichiers et services.

— Cmdlets principales : `Get-`, `Set-`, `New-`, `Remove-`.

—

14 Active Directory et GPO

14.1 Lister et gérer les utilisateurs

```
Get-ADUser -Filter * | Select-Object Name, Enabled
New-ADUser -Name "Jean Dupont" -SamAccountName jdupont ‘
 -AccountPassword (Read-Host -AsSecureString) -Enabled $true
```

14.2 Créer et lier une GPO

```
New-GPO -Name "GPO_Secu_BitLocker"
New-GPLink -Name "GPO_Secu_BitLocker" -Target "OU=Postes,DC=entreprise,DC=local"
```

—

15 Logs et supervision

15.1 Lire et exporter les événements

```
Get-EventLog -LogName System -Newest 20
Get-EventLog -LogName Security -Newest 50 | Export-Csv "C:\Logs\Security.csv"
```

15.2 Supervision en boucle

```
while ($true) {
   Get-EventLog -LogName System -EntryType Error -Newest 1
   Start-Sleep -Seconds 30
}
```

—

16 Sécurité avancée

16.1 BitLocker

```
Get-BitLockerVolume | Select-Object MountPoint, ProtectionStatus, EncryptionMethod
Enable-BitLocker -MountPoint "C:" -TpmProtector -UsedSpaceOnly -EncryptionMethod XtsAes2
```

16.2 Permissions NTFS

```
$acl = Get-Acl "C:\Data"
$rule = New-Object System.Security.AccessControl.FileSystemAccessRule("Utilisateurs","Re
$acl.SetAccessRule($rule)
Set-Acl "C:\Data" $acl
```

—

17 Administration distante avec PowerShell

17.1 Activation de WinRM (sur cible)

```
Enable-PSRemoting -Force
```

Cette commande configure le service **WinRM** (Windows Remote Management), ouvre le firewall et permet d'utiliser PowerShell à distance.

17.2 Exécution d'une commande distante

```
Invoke-Command -ComputerName SRV01 -ScriptBlock { Get-Service }
```

17.3 Connexion interactive distante

```
Enter-PSSession -ComputerName SRV01 -Credential domaine\admin
```

Cette commande ouvre une session interactive PowerShell sur le serveur `SRV01`.

17.4 Scripts multi-serveurs

```
$servers = @("SRV01","SRV02","SRV03")
Invoke-Command -ComputerName $servers -ScriptBlock {
    Get-Service -Name Spooler
}
```

17.5 Exécution parallèle avec `Invoke-Command`

```
Invoke-Command -ComputerName $servers -ScriptBlock {
    Get-EventLog -LogName System -Newest 5
} -AsJob
```

17.6 Utilisation de fichiers d'inventaire

```
# Lire une liste de serveurs depuis un fichier
$servers = Get-Content "C:\Scripts\serveurs.txt"

# Vérifier l'espace disque à distance
Invoke-Command -ComputerName $servers -ScriptBlock {
    Get-PSDrive C
}
```

—

18 TP pratiques administration distante

18.1 TP7 — Vérification de services sur plusieurs serveurs

— Créez une liste de serveurs dans `serveurs.txt`.
— Écrire un script qui affiche l'état du service `Spooler` sur chaque serveur.

18.2 TP8 — Audit espace disque distant

— Sur plusieurs serveurs, collectez l'espace libre du disque `C:`.
— Exportez les résultats dans un CSV.

18.3 TP9 — Supervision centralisée

— Utiliser `Invoke-Command` pour récupérer les 10 derniers événements critiques du journal Système sur tous les serveurs.
— Générer un rapport consolidé.
—

19 Conclusion

À présent, vous maîtrisez :
— La gestion locale et distante des systèmes Windows via PowerShell.
— Active Directory et GPO en ligne de commande.
— L'automatisation de la supervision et de la sécurité.
— L'administration multi-serveurs grâce à WinRM et PowerShell Remoting.
Ces compétences vous permettent d'automatiser entièrement la gestion d'un parc Windows en environnement professionnel.

20 Gestion d'API REST en PowerShell

Objectifs

À la fin de cette section, vous saurez :
— Interroger des API REST avec `Invoke-RestMethod` et `Invoke-WebRequest`.
— Gérer l'authentification (Basic, Bearer/OAuth2), les en-têtes et le JSON.
— Traiter la pagination, les erreurs, la limitation de débit (429) et les reprises.
— Envoyer des données (POST, PATCH, PUT, DELETE), y compris multipart/form-data.
— Télécharger / téléverser des fichiers via HTTP(S).

20.1 Cmdlets principales

— **Invoke-RestMethod (IRM)** : JSON ↔ objets PowerShell, idéal pour API REST.
— **Invoke-WebRequest (IWR)** : accès bas niveau (en-têtes complets, contenu brut), utile pour téléchargements.

20.2 Requêtes simples (GET)

```
# GET JSON --> objets PowerShell
$resp = Invoke-RestMethod -Uri "https://api.example.com/v1/items"
$resp.items | Select-Object id, name, createdAt
```

20.3 En-têtes et query string

```
$headers = @{ "X-Api-Key" = "VOTRE_CLE_API"; "Accept" = "application/json" }
$params  = @{ category = "servers"; limit = 50 }
$uri     = "https://api.example.com/v1/assets"

$resp = Invoke-RestMethod -Uri $uri -Headers $headers -Body $params -Method Get
```

20.4 Envoi de données JSON (POST / PATCH)

```
$payload = @{
  name = "SRV-PROD-01"
  role = "web"
  tags = @("prod","frontend")
} | ConvertTo-Json -Depth 5

$headers = @{
  "Authorization" = "Bearer VOTRE_JETON"
  "Content-Type"  = "application/json"
}

$resp = Invoke-RestMethod -Uri "https://api.example.com/v1/servers" `
  -Method Post -Headers $headers -Body $payload
```

20.5 Authentification

Basic Auth

```
$pair    = "user:motdepasse"
$b64     = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes($pair))
$headers = @{ Authorization = "Basic $b64" }
Invoke-RestMethod -Uri $uri -Headers $headers
```

Bearer / OAuth2 (flux client credentials)

```
$tokenResp = Invoke-RestMethod -Method Post -Uri "https://login.example.com/oauth2/token
  -Body @{
    client_id     = "VOTRE_CLIENT_ID"
    client_secret = "VOTRE_SECRET"
    grant_type    = "client_credentials"
    scope         = "api.read api.write"
  }
$headers = @{ Authorization = "Bearer " + $tokenResp.access_token }
Invoke-RestMethod -Uri "https://api.example.com/v1/me" -Headers $headers
```

20.6 Pagination

Pagination par page/offset

```
$page = 1; $all = @()
do {
  $resp = Invoke-RestMethod -Uri "https://api.example.com/v1/items?page=$page&per_page=1
```

```
  $all  += $resp.items
  $page++
} while ($resp.items.Count -gt 0)
```

Pagination par lien next

```
$uri = "https://api.example.com/v1/items"
$all = @()
while ($uri) {
  $r    = Invoke-WebRequest -Uri $uri -Headers $headers
  $json = $r.Content | ConvertFrom-Json
  $all += $json.items
  # Exemple: Link: <https://.../items?page=3>; rel="next"
  $next = $r.Headers.Link -match 'rel="next"' ? ($r.Headers.Link -split ';')[0] : $null
  $uri  = $next -replace '[<> ]',''
}
```

20.7 Gestion d'erreurs & reprise (429/5xx) avec backoff

```
function Invoke-RestSafe {
  param(
    [Parameter(Mandatory)] [string]$Uri,
    [ValidateSet('Get','Post','Put','Patch','Delete')] [string]$Method = 'Get',
    [hashtable]$Headers, $Body, [int]$MaxRetry = 5
  )
  $delay = 1
  for ($i=1; $i -le $MaxRetry; $i++) {
    try {
      return Invoke-RestMethod -Uri $Uri -Method $Method -Headers $Headers -Body $Body `
             -ContentType ($Headers['Content-Type'] ?? 'application/json')
    } catch {
      $status = $_.Exception.Response.StatusCode.value__
      if ($status -eq 429 -or ($status -ge 500 -and $status -lt 600)) {
        $retryAfter = $_.Exception.Response.Headers['Retry-After']
        $sleep = [int]($retryAfter ? $retryAfter : $delay)
        Start-Sleep -Seconds $sleep
        $delay = [Math]::Min($delay * 2, 60)
      } else { throw }
    }
  }
  throw "Échec après $MaxRetry tentatives: $Uri"
}
```

20.8 Téléchargements / uploads

Télécharger un fichier

```
Invoke-WebRequest -Uri "https://example.com/report.pdf" -OutFile "C:\Temp\report.pdf"
```

Upload multipart/form-data

```
# PowerShell 7+ : -Form
$form = @{
  "metadata" = '{"ticket":"INC123"}'
  "file"     = Get-Item "C:\Temp\log.zip"
}
Invoke-WebRequest -Uri "https://api.example.com/v1/upload" -Method Post -Form $form -Out
```

20.9 Cas pratiques (modèles)

GitHub API (GET publiques)

```
$uri = "https://api.github.com/repos/microsoft/PowerShell/releases?per_page=5"
$headers = @{ "User-Agent" = "PS-Training" ; "Accept" = "application/vnd.github+json" }
$releases = Invoke-RestMethod -Uri $uri -Headers $headers
$releases | Select-Object name, tag_name, published_at
```

Microsoft Graph (liste des utilisateurs) — Application (client credentials)

```
# 1) Récupérer le token (exemple Entra ID / MS Graph)
$token = Invoke-RestMethod -Method Post `
  -Uri "https://login.microsoftonline.com/VOTRE_TENANT_ID/oauth2/v2.0/token" `
  -Body @{
    client_id     = "VOTRE_CLIENT_ID"
    client_secret = "VOTRE_SECRET"
    scope         = "https://graph.microsoft.com/.default"
    grant_type    = "client_credentials"
  }

# 2) Appeler Graph
$h = @{ Authorization = "Bearer " + $token.access_token }
$users = Invoke-RestMethod -Uri "https://graph.microsoft.com/v1.0/users?$top=25" -Header
$users.value | Select-Object displayName, userPrincipalName, accountEnabled
```

20.10 Proxies, TLS et certificats

```
# Utiliser un proxy
[System.Net.WebRequest]::DefaultWebProxy = New-Object System.Net.WebProxy("http://proxy:

# Autoriser (temporairement) un certificat auto-signé (non recommandé en prod)
add-type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertsPolicy : ICertificatePolicy {
  public bool CheckValidationResult(ServicePoint srvPoint, X509Certificate certificate,
    WebRequest request, int certificateProblem) { return true; }
}
"@
[System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy
```

Attention : évitez de désactiver la validation TLS en production. Préférez déployer une AC interne et des certificats valides.

20.11 Structure recommandée (module REST)

```
# Fichier: Company.ApiTools.psm1
function New-ApiHeaders { param([string]$Token) @{ Authorization="Bearer $Token"; Accept
function Get-ApiPaged {
  param([string]$BaseUri,[hashtable]$Headers)
  $all=@(); $uri=$BaseUri
  while ($uri) {
    $r = Invoke-WebRequest -Uri $uri -Headers $Headers
    $j = $r.Content | ConvertFrom-Json
    $all += $j.value ?? $j.items
    $uri = ($r.Headers.Link -match 'rel="next"') ? (($r.Headers.Link -split ';')[0] -rep
  }
  return $all
}
Export-ModuleMember -Function *
```

20.12 TP pratiques REST

20.12.1 TP14 — GET + filtrage + export

- Interrogez une API publique (ex. GitHub) pour récupérer les 50 derniers dépôts d'une organisation.
- Filtrez les champs (nom, date de création, nombre d'étoiles) et exportez en CSV.

20.12.2 TP15 — POST JSON avec reprise

- Créez un script qui crée un objet via `POST` sur une API de test (utilisez un bac à sable).
- Intégrez `Invoke-RestSafe` pour gérer 429/5xx et journalisez les tentatives.

20.12.3 TP16 — Pagination & rapport

- Parcourez toutes les pages d'une API paginée et générez un rapport agrégé (compte par statut, top 10 par métrique).

20.12.4 TP17 — Upload multipart et ticketing

- Envoyez un fichier log compressé (`.zip`) en multipart sur une API de ticketing (sandbox).
- Vérifiez la réponse et extrayez l'ID du ticket créé.

20.13 Bonnes pratiques REST

- Toujours définir `Accept` et `Content-Type` appropriés.
- Utiliser `ConvertTo-Json -Depth N` (souvent $N \geq 5$) pour les payloads imbriqués.
- Centraliser l'authentification et les en-têtes dans des fonctions utilitaires.
- Gérer les erreurs avec `try/catch`, logs, et backoff exponentiel.
- Ne jamais commiter de secrets : utiliser variables d'environnement, coffre-fort (ex. Windows Credential Manager, Azure Key Vault).