UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Powershell

Sommaire

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction et bases de PowerShell

Introduction et bases de PowerShell

Introduction

1. Historique de PowerShell

- **Origines :**
  - Créé par Microsoft pour fournir un shell en ligne de commande et un langage de script destiné à l'administration système.
  - Introduit pour la première fois en **2006** sous le nom **Monad** (nom de code).

Introduction et bases de PowerShell

Introduction

1. Historique de PowerShell

- **Évolutions :**
  - **PowerShell 1.0 (2006)** : Sortie initiale, intégrée à Windows Server 2008 et Windows Vista.
  - **PowerShell 2.0 (2009)** : Ajout de nombreuses fonctionnalités, dont PowerShell Remoting via WinRM.
  - **PowerShell 3.0 (2012)** : Intégré à Windows Server 2012 et Windows 8, avec une meilleure prise en charge des modules.
  - **PowerShell 4.0 (2013)** : Introduction de Desired State Configuration (DSC).
  - **PowerShell 5.0/5.1 (2016)** : Intégration avec Windows Management Framework (WMF), prise en charge des classes et des nouveaux cmdlets.
  - **PowerShell Core 6.x (2018)** : Passage à une version multiplateforme (Windows, Linux, macOS), basé sur .NET Core.
  - **PowerShell 7.x (2020)** : Version actuelle, entièrement multiplateforme, basée sur .NET 5/6, avec de meilleures performances et compatibilité accrue avec les scripts Windows PowerShell.

Version	Année de sortie	Nouveautés clés	Compatibilité
PowerShell 1.0	2006	Cmdlets de base, pipelines	Windows uniquement
PowerShell 2.0	2009	Remoting, modules	Windows uniquement
PowerShell 3.0	2012	Améliorations des workflows, tâches planifiées	Windows uniquement
PowerShell 5.1	2016	Classes, gestion des paquets, DSC amélioré	Windows uniquement
PowerShell Core 6.x	2018	Multiplateforme, open source	Windows, macOS, Linux
PowerShell 7.x	2020+	Multiplateforme, basée sur .NET 5/6	Windows, macOS, Linux

Introduction et bases de PowerShell

Introduction

3. Cas d'usage de PowerShell

PowerShell est conçu pour répondre aux besoins d'administration et d'automatisation dans des environnements Windows et multiplateformes. Voici ses principaux cas d'usage :

1. **Administration système et automatisation :**
   - Gestion des utilisateurs et des groupes locaux et Active Directory.
   - Gestion des processus, services, et planification des tâches.
   - Surveillance et analyse des journaux d'événements.
2. **Gestion des fichiers et des dossiers :**
   - Création, copie, déplacement et suppression de fichiers et dossiers.
   - Recherche avancée dans les fichiers grâce aux filtres.
   - Analyse et transformation des données (CSV, JSON, XML).
3. **Gestion des infrastructures réseau :**
   - Configuration et surveillance des interfaces réseau.
   - Tests de connectivité et résolution DNS.
   - Automatisation des tâches réseau (paramétrage de pare-feu, VPN, etc.).

Introduction et bases de PowerShell

Introduction

3. Cas d'usage de PowerShell

4. **Déploiement et configuration :**
   - Déploiement de logiciels via des scripts automatisés.
   - Configuration des serveurs avec **Desired State Configuration (DSC)**.
   - Automatisation des mises à jour et correctifs.
5. **Gestion de la sécurité :**
   - Audit des permissions sur les fichiers et dossiers.
   - Gestion des certificats numériques.
   - Création de scripts sécurisés et gestion des credentials.
6. **Intégration et gestion de Cloud :**
   - Interaction avec Azure, AWS, ou Google Cloud à travers leurs modules spécifiques.
   - Gestion des machines virtuelles et ressources cloud.
7. **Scripting multiplateforme :**
   - Grâce à PowerShell Core et PowerShell 7.x, gestion des systèmes Windows, Linux et macOS avec le même outil.

Introduction et bases de PowerShell

Introduction

Avantages de PowerShell

- Basé sur **.NET Framework/Core** : permet de manipuler des objets directement au lieu de simples textes.
- **Extensibilité :** supporte des modules personnalisés et des API.
- **Interopérabilité :** fonctionne sur plusieurs plateformes et peut interagir avec d'autres outils comme Ansible, Terraform, etc.
- **Communauté active :** riche en modules préexistants sur des plateformes comme le **PowerShell Gallery**.

Introduction et bases de PowerShell

Introduction

Composants PowerShell

Composant	Description	Exemple
Shell interactif	Interface pour exécuter des commandes et scripts.	`Write-Host "Hello, PowerShell!"`
Cmdlets	Commandes de base suivant le modèle `Verbe-Nom`.	`Get-Process`, `Set-Item`
Pipelines	Chaînage de commandes, en passant les objets entre elles.	`Get-Process
Objets	Sortie des cmdlets sous forme d'objets .NET avec propriétés et méthodes.	`$process = Get-Process; $process[0].Name`

Introduction et bases de PowerShell

Introduction

Composants PowerShell

Composant	Description	Exemple
Fichiers de script	Scripts PowerShell (.ps1) contenant des séquences de commandes.	`Write-Host "Bonjour !"` (dans un fichier `Example.ps1`)
Modules	Collections de cmdlets, fonctions et scripts pour des tâches spécifiques.	`Import-Module Az`
Providers	Accès uniforme aux magasins de données (fichiers, registre, etc.).	`Get-ChildItem -Path HKLM:\Software`
Variables	Stockage de données ou d'objets avec `$NomVariable`.	`$name = "Alice"; Write-Host $name`

Introduction et bases de PowerShell

Composant	Description	Exemple
Fonctions	Groupes de commandes réutilisables, avec paramètres optionnels.	`function Greet { param([string]$Name) Write-Host "Hello, $Name!" }`
Gestion des erreurs	Gestion structurée des erreurs avec `try/catch/finally` ou `Get-Error`.	`try { Get-Item "InvalidPath" } catch { Write-Host "Erreur : $($_.Exception.Message)" }`
Remote PowerShell	Exécution de commandes sur des machines distantes via PowerShell Remoting.	`Invoke-Command -ComputerName Server01 -ScriptBlock { Get-Service }`
Integrated Scripting Environment (ISE)	Environnement graphique pour écrire et tester des scripts (remplacé par VS Code).	Utilisation d'outils modernes comme Visual Studio Code pour éditer et exécuter des scripts.

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

1. Vérification de la version PowerShell existante

1. **Ouvrir une session PowerShell :**
   - Sous Windows : Rechercher `PowerShell` dans le menu Démarrer et cliquer sur **Windows PowerShell** ou **PowerShell 7**.
   - Sous macOS ou Linux : Ouvrir un terminal.
2. **Commande pour vérifier la version :**

```
$PSVersionTable
```

   - La valeur de `PSVersion` indique la version installée.
   - Si aucune version n'est trouvée, PowerShell n'est pas installé.

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

2. Installation de PowerShell sur différentes plateformes

a. Installation sur Windows

- **Windows PowerShell (version classique)** : Déjà inclus dans Windows 7, 8.1, 10, 11, et Windows Server.
- **PowerShell 7+ (version multiplateforme)** :
  1. Télécharger l'installateur depuis la page officielle : https://github.com/PowerShell/PowerShell/releases.
  2. Choisir le package **.msi** correspondant à votre architecture (x64 ou x86).
  3. Suivre l'assistant d'installation.
- **Via Winget (gestionnaire de paquets Windows 10/11)** :

```
winget install --id Microsoft.Powershell --source winget
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

2. Installation de PowerShell sur différentes plateformes

b. Installation sur macOS

1. Ouvrir le terminal.
2. Installer PowerShell via **Homebrew** :

```
brew install --cask powershell
```

3. Lancer PowerShell :

```
pwsh
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

2. Installation de PowerShell sur différentes plateformes

c. Installation sur Linux

1. **Ajouter le dépôt officiel :**
   - Pour Ubuntu/Debian :

```
sudo apt-get update
sudo apt-get install -y wget apt-transport-https software-properties-common
wget -q "https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb" -O packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
```

   - Pour CentOS/RedHat :

```
sudo dnf install -y https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

2. Installation de PowerShell sur différentes plateformes

c. Installation sur Linux

2. **Installer PowerShell :**
   - Pour Ubuntu/Debian :

```
sudo apt-get update
sudo apt-get install -y powershell
```

   - Pour CentOS/RedHat :

```
sudo dnf install -y powershell
```

3. Lancer PowerShell :

```
pwsh
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

3. Configuration de l'environnement PowerShell

a. Modifier le profil utilisateur PowerShell

1. **Vérifier l'existence du profil utilisateur :**

```
Test-Path $PROFILE
```

   - Si `False`, créer le fichier de profil :

```
New-Item -Path $PROFILE -Type File -Force
```

2. **Éditer le fichier de profil :**

```
notepad $PROFILE
```

   - Exemple de contenu pour personnaliser l'environnement :

```
# Couleur de la console
$host.UI.RawUI.BackgroundColor = "DarkBlue"
$host.UI.RawUI.ForegroundColor = "White"
Clear-Host

# Alias personnalisés
Set-Alias ll Get-ChildItem
Set-Alias cls Clear-Host
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

3. Configuration de l'environnement PowerShell

b. Activer l'exécution de scripts

- Par défaut, PowerShell limite l'exécution de scripts pour des raisons de sécurité.

1. **Vérifier la politique actuelle :**

```
Get-ExecutionPolicy
```

2. **Modifier la politique (si nécessaire) :**
   - Pour permettre tous les scripts locaux :

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

   - Pour désactiver les restrictions :

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

   - **Note :** Cette commande nécessite des droits administratifs.

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

3. Configuration de l'environnement PowerShell

c. Installer des modules supplémentaires

1. Utiliser le PowerShell Gallery pour installer des modules :

```
Install-Module -Name Az -AllowClobber -Scope CurrentUser
```

2. Vérifier les modules installés :

```
Get-InstalledModule
```

d. Tester l'installation

- Exécuter une commande pour valider le bon fonctionnement :

```
Get-Process
```

Introduction et bases de PowerShell

Installation et configuration de l'environnement PowerShell

4. Personnalisation avancée (facultatif)

- Installer un thème personnalisé pour Visual Studio Code avec PowerShell Extension.
- Ajouter des outils tiers comme **Oh-My-Posh** pour un prompt stylé :

```
Install-Module -Name oh-my-posh -Scope CurrentUser
```

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

1. PowerShell ISE (Integrated Scripting Environment)

Présentation :

- PowerShell ISE est un environnement graphique fourni par Microsoft pour éditer, tester et exécuter des scripts PowerShell.
- Disponible par défaut sur Windows, jusqu'à PowerShell 5.1 (il n'est pas inclus avec PowerShell Core ou PowerShell 7).

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

1. PowerShell ISE (Integrated Scripting Environment)

Caractéristiques principales :

- Interface intuitive et intégrée pour l'écriture et le débogage de scripts PowerShell.
- Support du multi-panneaux :
  - **Panneau Script** : Éditeur pour écrire des scripts PowerShell.
  - **Panneau Console** : Exécution des commandes et retour des résultats.
- Fonctionnalités d'aide à l'écriture :
  - Coloration syntaxique.
  - Intellisense pour la complétion automatique des cmdlets et des paramètres.
- Outils intégrés :
  - Aide interactive (`Ctrl + J` pour les modèles).
  - Débogueur intégré avec points d'arrêt.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

1. PowerShell ISE (Integrated Scripting Environment)

Naviguer dans l'interface PowerShell ISE :

1. **Ouvrir PowerShell ISE :**
   - Rechercher `PowerShell ISE` dans le menu Démarrer.
   - Utiliser la commande suivante dans une console PowerShell classique :

     ```
     ise
     ```

2. **Structure de l'interface :**
   - **Barre d'outils** : Boutons pour exécuter, arrêter, déboguer ou enregistrer les scripts.
   - **Panneau Script** (en haut) : Éditeur de texte pour les scripts `.ps1`.
   - **Panneau Console** (en bas) : Exécute les commandes interactives ou teste des extraits de script.
   - **Panneau des variables** (facultatif) : Affiche les variables en cours d'utilisation.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

1. PowerShell ISE (Integrated Scripting Environment)

Naviguer dans l'interface PowerShell ISE :

3. **Exécution d’un script :**
    - Charger un script PowerShell (`.ps1`) dans le panneau script.
    - Cliquer sur **Run Script (F5)** ou sur **Run Selection (F8)** pour exécuter une sélection.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

1. PowerShell ISE (Integrated Scripting Environment)

Avantages et limitations :

- **Avantages :**
  - Simple à utiliser pour débuter en PowerShell.
  - Débogueur intégré avec points d’arrêt.
  - Ne nécessite pas de configuration supplémentaire.
- **Limitations :**
  - Ne prend pas en charge PowerShell Core ou 7+.
  - Pas aussi performant et extensible que Visual Studio Code.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

2. Visual Studio Code avec l'extension PowerShell

Présentation :

- Visual Studio Code (VS Code) est un éditeur de code moderne et multiplateforme.
- Avec l'extension PowerShell, il devient un environnement puissant pour éditer, déboguer et exécuter des scripts PowerShell.
- Compatible avec PowerShell 5.1, Core et 7+.

Installation de Visual Studio Code et de l'extension PowerShell :

1. **Télécharger et installer Visual Studio Code :**
    - Depuis le site officiel : https://code.visualstudio.com/.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

2. Visual Studio Code avec l'extension PowerShell

Installation de Visual Studio Code et de l'extension PowerShell :

2. **Installer l'extension PowerShell :**
   - Ouvrir VS Code.
   - Aller dans le **Marketplace** (icône Extensions ou `Ctrl+Shift+X`).
   - Rechercher `PowerShell` et cliquer sur **Install**.
   - Extension officielle : `PowerShell by Microsoft`.
3. **Configurer PowerShell comme terminal par défaut :**
   - Ouvrir les **Paramètres** de VS Code (`Ctrl+,`).
   - Ajouter ou modifier dans `settings.json` :

```
"terminal.integrated.defaultProfile.windows": "PowerShell"
```

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

2. Visual Studio Code avec l'extension PowerShell

Naviguer dans l'interface Visual Studio Code :

1. **Structure de l'interface :**
   - **Barre latérale gauche** : Contient les fonctionnalités principales (explorateur de fichiers, recherche, contrôle de version, extensions).
   - **Éditeur principal** : Fenêtre pour éditer les fichiers PowerShell (`.ps1`).
   - **Terminal intégré** (en bas) : Pour exécuter des commandes PowerShell en ligne.
2. **Écrire et exécuter un script PowerShell :**
   - Créer un nouveau fichier PowerShell : `File > New File` et sauvegarder avec l'extension `.ps1`.
   - Taper votre script dans l'éditeur.
   - Exécuter le script :
     - Cliquer sur **Run Script** (triangle vert en haut à droite).
     - Utiliser le raccourci `F5`.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

2. Visual Studio Code avec l'extension PowerShell

Naviguer dans l'interface Visual Studio Code :

3. **Déboguer un script :**
    - Ajouter un point d’arrêt en cliquant dans la marge à gauche de l'éditeur.
    - Lancer le débogage (`F5`).
    - Utiliser les commandes du débogueur pour avancer dans le script (Step Into, Step Over).

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

2. Visual Studio Code avec l'extension PowerShell

Fonctionnalités supplémentaires avec l'extension PowerShell :

- Coloration syntaxique avancée.
- Suggestions et auto-complétion pour les cmdlets, paramètres et variables.
- Intégration avec Git pour le contrôle de version.
- Débogueur interactif amélioré.

Avantages et limitations :

- **Avantages :**
  - Multiplateforme et extensible.
  - Interface moderne avec support pour les extensions.
  - Compatible avec toutes les versions de PowerShell.
- **Limitations :**
  - Nécessite une configuration initiale pour les débutants.

Introduction et bases de PowerShell

Compréhension de l'interface PowerShell ISE et de Visual Studio Code

Fonctionnalité	PowerShell ISE	Visual Studio Code
Compatibilité	Windows uniquement (PowerShell 5.1)	Multiplateforme (Windows, Linux, macOS)
Performances	Moins rapide	Très rapide
Débogage	Intégré	Débogueur amélioré
Extensibilité	Limité	Nombreuses extensions disponibles
Interface utilisateur	Simple	Moderne et personnalisable
Mises à jour	Arrêtées	Actives

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

1. Cmdlets (Command-lets)

Qu'est-ce qu'une cmdlet ?

- Une cmdlet est une commande intégrée dans PowerShell pour effectuer des actions spécifiques.
- Elle suit le schéma de **verbe-nom** (par exemple, `Get-Process`, `Set-Item`).

Caractéristiques :

- Les cmdlets sont des objets .NET, ce qui permet de manipuler directement leurs propriétés et méthodes.
- Elles sont différentes des commandes classiques des autres shells (comme Bash), car elles retournent des objets, pas du texte.

Structure générale d'une cmdlet :

```
<Verbe>-<Nom> [-Paramètres] [Valeur(s)]
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

1. Cmdlets (Command-lets)

Cmdlet	Description
`Get-Help`	Affiche de l'aide sur une cmdlet ou un sujet.
`Get-Command`	Liste toutes les cmdlets disponibles.
`Get-Process`	Liste les processus en cours.
`Set-Location`	Change le répertoire de travail.
`New-Item`	Crée un fichier ou un dossier.

Exemple pratique :

```
# Récupère une liste des processus
Get-Process
# Affiche des détails sur la cmdlet Get-Process
Get-Help Get-Process
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

2. Pipelines

Qu'est-ce qu'un pipeline ?

- Le pipeline (`|`) est un mécanisme pour envoyer la sortie d'une cmdlet en entrée d'une autre.
- C'est une manière de **chaîner plusieurs commandes** pour effectuer des traitements complexes.

Fonctionnement du pipeline :

1. La cmdlet produit une sortie sous forme d'objet(s).
2. Ces objets sont envoyés directement à la cmdlet suivante comme entrée.

Exemples simples :

1. **Lister les processus et filtrer par nom :**

```
Get-Process | Where-Object {$_.Name -eq "notepad"}
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

2. Pipelines

Exemples simples :

2. **Afficher les fichiers d'un dossier triés par taille :**

```
Get-ChildItem | Sort-Object Length
```

3. **Exporter des données vers un fichier CSV :**

```
Get-Process | Export-Csv -Path processes.csv -NoTypeInformation
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

2. Pipelines

Pipeline avec plusieurs étapes :

```
Get-Process |
    Where-Object {$_.CPU -gt 10} | # Filtre les processus avec une consommation CPU > 10
    Sort-Object CPU -Descending | # Trie par utilisation CPU décroissante
    Select-Object -First 5          # Sélectionne les 5 premiers résultats
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

3. Objets en PowerShell

Pourquoi les objets sont importants ?

- PowerShell traite tout comme un **objet**.
- Contrairement à d'autres shells où les commandes retournent du texte brut, PowerShell retourne des objets .NET.

Structure des objets :

Un objet possède :

- **Propriétés** : Des attributs contenant des informations sur l'objet.
- **Méthodes** : Des actions ou comportements que l'objet peut effectuer.

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

3. Objets en PowerShell

Exploration des objets :

1. **Lister les propriétés et méthodes d'un objet :**

```
Get-Process | Get-Member
```

Exemple de sortie pour `Get-Process` :

```
Name                       MemberType   Definition
----                       ----------   ----------
CPU                        Property     System.Double CPU {get;}
StartTime                  Property     System.DateTime StartTime {get;}
Kill                       Method       void Kill()
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

3. Objets en PowerShell

Exploration des objets :

2. **Accéder aux propriétés d'un objet :**

```
Get-Process | Select-Object Name, CPU
```

3. **Appeler une méthode :**

```
# Terminer un processus
Get-Process notepad | ForEach-Object { $_.Kill() }
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

3. Objets en PowerShell

Manipulation avancée des objets :

- **Filtrer les objets :**

```
Get-Process | Where-Object {$_.WorkingSet -gt 500MB}
```

- **Modifier les propriétés d'un objet :**

```
$service = Get-Service | Select-Object -First 1
$service.Status = "Stopped"  # Modification d'une propriété
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

4. Combinaison cmdlets, pipelines et objets

Exemple pratique 1 : Filtrer les fichiers d'un répertoire

```
Get-ChildItem -Path C:\Windows -Recurse |
    Where-Object {$_.Length -gt 1MB} |
    Select-Object Name, Length
```

Exemple pratique 2 : Rapport sur les services

```
Get-Service |
    Where-Object {$_.Status -eq "Running"} |
    Select-Object Name, Status, StartType |
    Export-Csv -Path running-services.csv -NoTypeInformation
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

4. Combinaison cmdlets, pipelines et objets

Exemple pratique 3 : Trouver les 5 processus utilisant le plus de mémoire

```
Get-Process |
    Sort-Object -Property WorkingSet -Descending |
    Select-Object -First 5 Name, WorkingSet
```

Introduction et bases de PowerShell

Introduction à la syntaxe PowerShell : Cmdlets, Pipelines et Objets

5. Bonnes pratiques

1. **Utilisez `Get-Help` pour comprendre les cmdlets :**

```
Get-Help <Cmdlet> -Examples
```

2. **Utilisez le pipeline pour simplifier le traitement des données :**
   - Chaque étape traite un petit morceau du travail.
3. **Utilisez `Get-Member` pour explorer les propriétés/méthodes des objets.**

```
Get-Service | Get-Member
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les commandes de base

Les commandes de base

Cmdlets Fondamentaux en PowerShell

- PowerShell offre plusieurs cmdlets de base qui permettent de découvrir, d'apprendre et de maîtriser ses fonctionnalités.
- Ces cmdlets sont essentiels pour explorer l'environnement PowerShell et interagir avec celui-ci.

Les commandes de base

Type	Exemples de cmdlets
Gestion des fichiers	`Get-Content`, `Remove-Item`, `New-Item`
Gestion des processus	`Get-Process`, `Start-Process`
Gestion des services	`Get-Service`, `Start-Service`
Gestion des utilisateurs	`Get-LocalUser`, `New-LocalUser`
Réseau	`Test-Connection`, `Get-NetIPAddress`
Administration système	`Restart-Computer`, `Get-EventLog`
Gestion des données	`Import-Csv`, `ConvertTo-Json`
Gestion des modules	`Get-Module`, `Import-Module`
Développement	`New-Item`, `Invoke-Command`
Sécurité	`Get-Acl`, `ConvertTo-SecureString`

Les commandes de base

Cmdlets Fondamentaux en PowerShell

1. `Get-Command`

Description :

- Permet de **lister toutes les commandes disponibles** dans l'environnement PowerShell, y compris les cmdlets, alias, fonctions, scripts et applications externes.

Syntaxe :

```
Get-Command [-Name <Nom>] [-CommandType <Type>] [-Module <NomModule>]
```

Exemples :

1. **Lister toutes les commandes disponibles :**

   ```
   Get-Command
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

1. `Get-Command`

Exemples :

2. **Rechercher une commande spécifique :**

```
Get-Command -Name Get-Process
```

3. **Lister uniquement les cmdlets :**

```
Get-Command -CommandType Cmdlet
```

4. **Lister les commandes d'un module spécifique :**

```
Get-Command -Module Microsoft.PowerShell.Management
```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

2. `Get-Help`

Description :

- Fournit **des informations détaillées** sur une cmdlet, une commande ou un concept spécifique.

Syntaxe :

```
Get-Help <Cmdlet> [-Full] [-Examples] [-Online]
```

Exemples :

1. **Afficher une aide rapide pour une cmdlet :**

   ```
   Get-Help Get-Process
   ```

2. **Afficher des exemples d'utilisation :**

   ```
   Get-Help Get-Process -Examples
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

2. `Get-Help`

Exemples :

3. **Afficher des détails complets :**

```
Get-Help Get-Process -Full
```

4. **Ouvrir la documentation en ligne (si disponible) :**

```
Get-Help Get-Process -Online
```

Astuce : Si l'aide n'est pas à jour, vous pouvez la mettre à jour avec :

```
Update-Help
```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

3. `Get-Alias`

Description :

- Permet de **découvrir les alias** des cmdlets PowerShell. Les alias sont des raccourcis pour certaines cmdlets, souvent inspirés des commandes utilisées dans d'autres shells comme Bash.

Syntaxe :

```
Get-Alias [-Name <NomAlias>] [-Definition <Cmdlet>]
```

Exemples :

1. **Lister tous les alias disponibles :**

   ```
   Get-Alias
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

3. `Get-Alias`

Exemples :

2. **Trouver la cmdlet correspondant à un alias spécifique :**

```
Get-Alias -Name gci
```

3. **Trouver les alias associés à une cmdlet spécifique :**

```
Get-Alias -Definition Get-ChildItem
```

4. **Créer un alias personnalisé :**

```
Set-Alias -Name ll -Value Get-ChildItem
```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

4. `Get-Module`

Description :

- Fournit des informations sur les modules PowerShell disponibles et chargés.

Syntaxe :

```
Get-Module [-Name <NomModule>] [-ListAvailable]
```

Exemples :

1. **Lister les modules chargés actuellement :**

   ```
   Get-Module
   ```

2. **Lister tous les modules disponibles sur le système :**

   ```
   Get-Module -ListAvailable
   ```

3. **Charger un module spécifique :**

   ```
   Import-Module -Name ActiveDirectory
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

5. `Get-Process`

Description :

- Affiche les informations sur les processus en cours d'exécution.

Syntaxe :

```
Get-Process [-Name <NomProcessus>] [-Id <IDProcessus>] [-ComputerName <NomMachine>]
```

Exemples :

1. **Lister tous les processus en cours :**

   ```
   Get-Process
   ```

2. **Afficher les détails d'un processus spécifique :**

   ```
   Get-Process -Name notepad
   ```

3. **Terminer un processus spécifique (avec `Stop-Process`) :**

   ```
   Get-Process -Name notepad | Stop-Process
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

6. `Set-Location`

Description :

- Change le répertoire de travail courant.

Syntaxe :

```
Set-Location [-Path <Chemin>]
```

Exemples :

1. **Naviguer vers un répertoire spécifique :**

   ```
   Set-Location -Path C:\Windows
   ```

2. **Utiliser des alias pour changer de répertoire :**

   ```
   cd C:\Users
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

7. `Get-ChildItem`

Description :

- Liste le contenu d'un répertoire (équivalent de `ls` en Linux).

Syntaxe :

```
Get-ChildItem [-Path <Chemin>] [-Filter <Filtre>] [-Recurse]
```

Exemples :

1. **Lister les fichiers dans le répertoire courant :**

   ```
   Get-ChildItem
   ```

2. **Lister les fichiers avec un filtre :**

   ```
   Get-ChildItem -Filter *.txt
   ```

3. **Rechercher récursivement dans les sous-dossiers :**

   ```
   Get-ChildItem -Path C:\ -Recurse
   ```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

8. `Clear-Host`

Description :

- Efface l'affichage dans la console PowerShell (équivalent de `clear` en Linux).

Syntaxe :

```
Clear-Host
```

Exemple :

```
Clear-Host
```

Les commandes de base

Cmdlets Fondamentaux en PowerShell

Cmdlet	Description
`Get-Command`	Liste toutes les commandes disponibles.
`Get-Help`	Fournit des informations détaillées sur une cmdlet.
`Get-Alias`	Montre les alias associés aux cmdlets.
`Get-Module`	Affiche les modules disponibles ou chargés.
`Get-Process`	Liste les processus en cours.
`Set-Location`	Change le répertoire de travail courant.
`Get-ChildItem`	Affiche le contenu d'un répertoire.
`Clear-Host`	Efface l'écran de la console.

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

- PowerShell offre des cmdlets puissantes pour naviguer et manipuler le système de fichiers.
- Les principales cmdlets utilisées pour explorer et changer de répertoire sont `Get-ChildItem` et `Set-Location`.

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

1. `Get-ChildItem`

Description :

- Permet de lister le contenu d'un répertoire.
- Peut être utilisé pour rechercher des fichiers, des dossiers ou d'autres types d'éléments.
- Alias courants : `gci`, `ls`, `dir`.

Syntaxe :

```
Get-ChildItem [-Path <Chemin>] [-Filter <Filtre>] [-Recurse] [-File] [-Directory]
```

Paramètres clés :

- **`-Path`** : Spécifie le chemin d'accès au répertoire (par défaut, le répertoire courant).
- **`-Filter`** : Filtre les résultats par nom ou extension.
- **`-Recurse`** : Recherche récursive dans les sous-dossiers.
- **`-File`** : Liste uniquement les fichiers.
- **`-Directory`** : Liste uniquement les répertoires.

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

1. `Get-ChildItem`

Exemples pratiques :

1. **Lister le contenu du répertoire courant :**

```
Get-ChildItem
```

2. **Lister les fichiers dans un répertoire spécifique :**

```
Get-ChildItem -Path C:\Users
```

3. **Lister uniquement les fichiers d'un type spécifique (par exemple, `.txt`) :**

```
Get-ChildItem -Path C:\Documents -Filter *.txt
```

4. **Recherche récursive dans les sous-dossiers :**

```
Get-ChildItem -Path C:\Projects -Recurse
```

5. **Lister uniquement les dossiers :**

```
Get-ChildItem -Path C:\ -Directory
```

6. **Lister uniquement les fichiers :**

```
Get-ChildItem -Path C:\ -File
```

7. **Afficher des informations spécifiques (nom, taille) :**

```
Get-ChildItem -Path C:\Documents | Select-Object Name, Length
```

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

2. `Set-Location`

Description :

- Change le répertoire de travail courant dans PowerShell.
- Alias courants : `cd`, `chdir`.

Syntaxe :

```
Set-Location [-Path] <Chemin>
```

Paramètres clés :

- **`-Path`** : Spécifie le chemin d'accès au répertoire où se déplacer.

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

2. `Set-Location`

Exemples pratiques :

1. **Changer vers un répertoire spécifique :**

```
Set-Location -Path C:\Users
```

2. **Utiliser un alias pour naviguer :**

```
cd C:\Documents
```

3. **Retourner au répertoire précédent :**

```
Set-Location -Path -
```

4. **Aller directement au répertoire de l'utilisateur :**

```
Set-Location -Path $HOME
```

5. **Naviguer dans un chemin relatif :**
   - Si le répertoire courant est `C:\Users`, aller dans `Documents` :

```
cd Documents
```

6. **Afficher le répertoire courant :**

```
Get-Location
```

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

Combinaison des cmdlets :

Exemple pratique 1 : Lister tous les fichiers `.log` d'un répertoire spécifique et changer de répertoire

```
Get-ChildItem -Path C:\Logs -Filter *.log
Set-Location -Path C:\Logs
```

Exemple pratique 2 : Recherche récursive et navigation

```
# Rechercher un fichier spécifique et naviguer dans son répertoire parent
Get-ChildItem -Path C:\Projects -Recurse -Filter myfile.txt |
    ForEach-Object { Set-Location -Path $_.DirectoryName }
```

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

Combinaison des cmdlets :

Exemple pratique 3 : Naviguer et afficher uniquement les fichiers importants

```
Set-Location -Path C:\Reports
Get-ChildItem -File | Where-Object { $_.Length -gt 1MB }
```

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

Alias utiles pour ces cmdlets :

Cmdlet	Alias	Description
`Get-ChildItem`	`gci`, `ls`, `dir`	Liste le contenu d'un répertoire.
`Set-Location`	`cd`, `chdir`	Change le répertoire courant.

Les commandes de base

Navigation dans le système de fichiers avec PowerShell

Commande	Description
`Get-ChildItem`	Liste les fichiers et dossiers dans le répertoire courant ou spécifié.
`Get-ChildItem -Recurse`	Recherche récursive dans les sous-dossiers.
`Get-ChildItem -File`	Liste uniquement les fichiers.
`Get-ChildItem -Directory`	Liste uniquement les répertoires.
`Set-Location <Chemin>`	Change le répertoire de travail courant.
`Get-Location`	Affiche le répertoire de travail actuel.
`cd ..`	Monte d'un niveau dans la hiérarchie des dossiers.
`Set-Location -Path -`	Revient au répertoire précédent.

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

- PowerShell offre un ensemble complet de cmdlets pour gérer efficacement les fichiers et les dossiers.
- Ces cmdlets permettent de créer, supprimer, copier, déplacer et renommer des fichiers ou des dossiers avec une syntaxe simple et intuitive.

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

1. Création de fichiers et dossiers

Cmdlet : `New-Item`

- Permet de créer des fichiers ou des dossiers.

Syntaxe :

```
New-Item -Path <Chemin> -ItemType <Type> -Name <Nom> [-Value <Contenu>]
```

Exemples :

1. **Créer un fichier vide :**

   ```
   New-Item -Path C:\Temp -Name example.txt -ItemType File
   ```

2. **Créer un fichier avec du contenu :**

   ```
   New-Item -Path C:\Temp -Name example.txt -ItemType File -Value "Hello, World!"
   ```

3. **Créer un dossier :**

   ```
   New-Item -Path C:\Temp -Name MyFolder -ItemType Directory
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

2. Suppression de fichiers et dossiers

Cmdlet : `Remove-Item`

- Supprime des fichiers ou des dossiers.

Syntaxe :

```
Remove-Item -Path <Chemin> [-Recurse] [-Force]
```

Exemples :

1. **Supprimer un fichier :**

   ```
   Remove-Item -Path C:\Temp\example.txt
   ```

2. **Supprimer un dossier vide :**

   ```
   Remove-Item -Path C:\Temp\MyFolder
   ```

3. **Supprimer un dossier avec tout son contenu :**

   ```
   Remove-Item -Path C:\Temp\MyFolder -Recurse
   ```

4. **Forcer la suppression (même si verrouillé) :**

   ```
   Remove-Item -Path C:\Temp\example.txt -Force
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

3. Copie de fichiers et dossiers

Cmdlet : `Copy-Item`

- Permet de copier des fichiers ou des dossiers.

Syntaxe :

```
Copy-Item -Path <Source> -Destination <Destination> [-Recurse] [-Force]
```

Exemples :

1. **Copier un fichier :**

   ```
   Copy-Item -Path C:\Temp\example.txt -Destination C:\Backup
   ```

2. **Copier un dossier et son contenu :**

   ```
   Copy-Item -Path C:\Temp\MyFolder -Destination C:\Backup -Recurse
   ```

3. **Forcer la copie (écrase les fichiers existants) :**

   ```
   Copy-Item -Path C:\Temp\example.txt -Destination C:\Backup -Force
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

4. Déplacement de fichiers et dossiers

Cmdlet : `Move-Item`

- Permet de déplacer ou renommer des fichiers ou des dossiers.

Syntaxe :

```
Move-Item -Path <Source> -Destination <Destination> [-Force]
```

Exemples :

1. **Déplacer un fichier :**

   ```
   Move-Item -Path C:\Temp\example.txt -Destination C:\Backup
   ```

2. **Déplacer un dossier et son contenu :**

   ```
   Move-Item -Path C:\Temp\MyFolder -Destination C:\Backup
   ```

3. **Renommer un fichier ou dossier lors du déplacement :**

   ```
   Move-Item -Path C:\Temp\example.txt -Destination C:\Temp\new_example.txt
   ```

4. **Forcer le déplacement :**

   ```
   Move-Item -Path C:\Temp\example.txt -Destination C:\Backup -Force
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

5. Renommage de fichiers et dossiers

Cmdlet : `Rename-Item`

- Renomme un fichier ou un dossier.

Syntaxe :

```
Rename-Item -Path <Chemin> -NewName <NouveauNom>
```

Exemples :

1. **Renommer un fichier :**

   ```
   Rename-Item -Path C:\Temp\example.txt -NewName new_example.txt
   ```

2. **Renommer un dossier :**

   ```
   Rename-Item -Path C:\Temp\MyFolder -NewName MyNewFolder
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

6. Vérification des fichiers et dossiers

Cmdlet : `Test-Path`

- Vérifie si un fichier ou un dossier existe.

Syntaxe :

```
Test-Path -Path <Chemin>
```

Exemples :

1. **Vérifier si un fichier existe :**

   ```
   Test-Path -Path C:\Temp\example.txt
   ```

2. **Vérifier si un dossier existe :**

   ```
   Test-Path -Path C:\Temp\MyFolder
   ```

3. **Condition basée sur l'existence d'un fichier :**

   ```
   if (Test-Path -Path C:\Temp\example.txt) {
       Write-Output "Le fichier existe."
   } else {
       Write-Output "Le fichier n'existe pas."
   }
   ```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

Exemple pratique : Gestion complète d'un fichier

1. **Créer un fichier avec du contenu, le copier, le déplacer, puis le supprimer :**

```
# Créer un fichier avec du contenu
New-Item -Path C:\Temp -Name example.txt -ItemType File -Value "Contenu initial"

# Copier le fichier
Copy-Item -Path C:\Temp\example.txt -Destination C:\Backup

# Déplacer et renommer le fichier
Move-Item -Path C:\Temp\example.txt -Destination C:\Temp\renamed_example.txt

# Supprimer le fichier
Remove-Item -Path C:\Temp\renamed_example.txt
```

Les commandes de base

Gestion des fichiers et des dossiers avec PowerShell

Résumé des cmdlets pour la gestion des fichiers et dossiers

Cmdlet	Description
`New-Item`	Crée un fichier ou un dossier.
`Remove-Item`	Supprime un fichier ou un dossier.
`Copy-Item`	Copie un fichier ou un dossier.
`Move-Item`	Déplace ou renomme un fichier ou un dossier.
`Rename-Item`	Renomme un fichier ou un dossier.
`Test-Path`	Vérifie l'existence d'un fichier ou d'un dossier.

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

- Le pipeline (|) est l'un des concepts fondamentaux de PowerShell.
- Il permet de chaîner plusieurs cmdlets en envoyant la **sortie d'une cmdlet** en **entrée d'une autre cmdlet**, facilitant ainsi des traitements complexes en une seule commande.

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

1. Comment fonctionne le pipeline en PowerShell ?

- PowerShell traite la sortie d'une cmdlet sous forme d'objets (pas du texte brut comme dans d'autres shells).
- Ces objets sont transmis au pipeline et deviennent l'entrée de la cmdlet suivante.
- Cela permet de manipuler directement les propriétés et méthodes des objets.

Syntaxe :

```
<Cmdlet1> | <Cmdlet2> | <Cmdlet3>
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

2. Exemples de base avec des pipelines

Lister, filtrer et trier des fichiers

1. **Lister les fichiers et dossiers, puis filtrer les fichiers uniquement :**

```
Get-ChildItem -Path C:\Temp | Where-Object {$_.PSIsContainer -eq $false}
```

2. **Lister les fichiers `.txt` triés par taille décroissante :**

```
Get-ChildItem -Path C:\Temp -Filter *.txt | Sort-Object -Property Length -Descending
```

3. **Lister les 5 plus gros fichiers dans un dossier :**

```
Get-ChildItem -Path C:\Temp | Sort-Object -Property Length -Descending | Select-Object -First 5
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

3. Manipulation des processus

Afficher les processus consommant beaucoup de CPU :

```
Get-Process | Where-Object {$_.CPU -gt 10}
```

Trier les processus par mémoire utilisée et afficher les 3 premiers :

```
Get-Process | Sort-Object -Property WorkingSet -Descending | Select-Object -First 3
```

Arrêter tous les processus nommés "notepad" :

```
Get-Process -Name notepad | Stop-Process
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

4. Exportation de données

Exporter la liste des services en cours d'exécution dans un fichier CSV :

```
Get-Service | Where-Object {$_.Status -eq "Running"} | Export-Csv -Path C:\Temp\Services.csv -NoTypeInformation
```

Exporter les 10 fichiers les plus récents dans un dossier :

```
Get-ChildItem -Path C:\Temp | Sort-Object -Property LastWriteTime -Descending | Select-Object -First 10 | Export-Csv -Path C:\Temp\RecentFiles.csv -NoTypeInformation
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

5. Combinaisons avancées

Rechercher des fichiers spécifiques et en afficher les détails :

```
Get-ChildItem -Path C:\Projects -Filter *.log -Recurse |
    Where-Object {$_.Length -gt 1MB} |
    Select-Object Name, Length, LastWriteTime
```

Lister les services en cours, trier par nom, et générer un rapport HTML :

```
Get-Service | Where-Object {$_.Status -eq "Running"} | Sort-Object -Property Name | ConvertTo-Html -Property Name, Status, StartType | Out-File -Path C:\Temp\ServicesReport.html
```

Analyser les journaux d'événements Windows :

```
Get-EventLog -LogName System | Where-Object {$_.EntryType -eq "Error"} | Select-Object Source, EventID, Message
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

6. Utilisation avec des scripts personnalisés

Transformation de données en chaîne avec des fonctions :

- Les pipelines fonctionnent aussi avec les **scripts personnalisés** et les **fonctions**.

1. **Créer une fonction pour filtrer des fichiers :**

```
function Get-LargeFiles {
    param(
        [string]$Path,
        [int]$Size
    )
    Get-ChildItem -Path $Path -Recurse | Where-Object {$_.Length -gt $Size}
}
```

2. **Utiliser la fonction dans un pipeline :**

```
Get-LargeFiles -Path C:\Temp -Size 100KB | Select-Object Name, Length
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

7. Étapes multiples avec des pipelines

Pipeline avec plusieurs étapes pour un traitement complexe :

```
# Étape 1 : Récupérer tous les fichiers
# Étape 2 : Filtrer les fichiers .txt
# Étape 3 : Trier par taille décroissante
# Étape 4 : Afficher les 3 plus gros fichiers
Get-ChildItem -Path C:\Temp |
    Where-Object {$_.Extension -eq ".txt"} |
    Sort-Object -Property Length -Descending |
    Select-Object -First 3
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

8. Astuces pour travailler avec les pipelines

Afficher les résultats intermédiaires avec `Write-Output` :

- Pour déboguer des pipelines complexes, utilisez `Write-Output` ou `Out-Host` pour afficher les résultats intermédiaires.

```
Get-ChildItem -Path C:\Temp |
    Write-Output |
    Where-Object {$_.Length -gt 1MB} |
    Select-Object Name, Length
```

Utiliser des commandes personnalisées dans les pipelines :

- Les cmdlets personnalisées, comme `ForEach-Object` ou `Select-Object`, peuvent être utilisées pour manipuler ou transformer les objets.

```
Get-ChildItem -Path C:\Temp |
    ForEach-Object { $_.Name.ToUpper() }
```

Les commandes de base

Utilisation des Pipelines pour Chaîner des Cmdlets

Résumé des Cmdlets courantes pour les pipelines

Cmdlet	Description
`Where-Object`	Filtrer les objets selon une condition.
`Sort-Object`	Trier les objets par une ou plusieurs propriétés.
`Select-Object`	Sélectionner des propriétés ou limiter les résultats.
`Export-Csv`	Exporter les données dans un fichier CSV.
`ConvertTo-Html`	Convertir les données en tableau HTML.
`Out-File`	Sauvegarder les résultats dans un fichier texte.
`ForEach-Object`	Exécuter un script sur chaque objet du pipeline.

Scripting : Variables et structures de données - Contrôle de flux

Variables et structures de données - Contrôle de flux

Variables et structures de données

1. Déclaration de Variables

- En PowerShell, les variables commencent toujours par un symbole `$`.
- Une variable peut être déclarée et initialisée sans type explicite (PowerShell est typé dynamiquement).

Syntaxe :

```
$NomVariable = Valeur
```

Exemples :

1. **Déclarer et assigner une valeur à une variable :**

```
$Nom = "Alice"
$Age = 30
```

2. **Afficher la valeur d'une variable :**

```
Write-Output $Nom
```

3. **Modifier la valeur d'une variable :**

```
$Age = 31
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

2. Variables automatiques

PowerShell fournit des **variables automatiques** qui stockent des informations spécifiques.

Variable	Description
`$PSVersionTable`	Informations sur la version PowerShell.
`$HOME`	Répertoire personnel de l'utilisateur.
`$Error`	Liste des dernières erreurs rencontrées.

Exemple :

```
Write-Output $HOME
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Types de Données en PowerShell

Types courants :

Type	Exemple
String	`"Hello, World"`
Int32	`42`
Double	`3.14`
Boolean	`$true`, `$false`
Array	`@(1, 2, 3)`
HashTable	`@{Key = "Value"}`
DateTime	`[datetime]::Now`

Variables et structures de données - Contrôle de flux

Variables et structures de données

Types de Données en PowerShell

Déclaration explicite d'un type :

```
[int]$Age = 25
[string]$Nom = "Alice"
```

Conversion de type :

1. **Convertir une chaîne en entier :**

```
$Nombre = [int]"42"
```

2. **Obtenir le type d'une variable :**

```
$Variable = 3.14
$Variable.GetType()
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Tableaux en PowerShell

Un **tableau** (Array) est une collection ordonnée d'éléments, accessible via des indices.

Déclaration d'un tableau :

1. **Tableau simple :**

```
$Nombres = @(1, 2, 3, 4, 5)
```

2. **Accéder aux éléments du tableau :**

```
$Nombres[0]  # Premier élément
$Nombres[-1] # Dernier élément
```

3. **Ajouter un élément à un tableau :**

```
$Nombres += 6
```

4. **Itérer sur un tableau :**

```
foreach ($Nombre in $Nombres) {
    Write-Output $Nombre
}
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Tableaux en PowerShell

Un **tableau** (Array) est une collection ordonnée d'éléments, accessible via des indices.

Méthodes de tableau :

- Obtenir le nombre d'éléments :

```
$Nombres.Count
```

- Filtrer les éléments :

```
$Nombres | Where-Object {$_ -gt 3}
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Tableaux Associatifs (Hash Tables)

Les **hash tables** sont des collections de paires clé-valeur.

Déclaration d'une hash table :

```
$Infos = @{
    Nom = "Alice"
    Age = 30
    Ville = "Paris"
}
```

Accéder à une valeur par clé :

```
$Infos["Nom"]  # Renvoie "Alice"
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Tableaux Associatifs (Hash Tables)

Ajouter ou modifier une entrée :

1. **Ajouter une clé :**

```
$Infos["Pays"] = "France"
```

2. **Modifier une valeur :**

```
$Infos["Ville"] = "Lyon"
```

Itérer sur une hash table :

```
foreach ($Cle in $Infos.Keys) {
    Write-Output "$Cle : $($Infos[$Cle])"
}
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Tableaux Associatifs (Hash Tables)

Exemple pratique :

```
$Serveurs = @{
    "Web" = "192.168.1.1"
    "DB" = "192.168.1.2"
    "Mail" = "192.168.1.3"
}

# Afficher tous les serveurs
foreach ($Role in $Serveurs.Keys) {
    Write-Output "$Role : $($Serveurs[$Role])"
}
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Exemple Complet : Variables, Tableaux et Hash Tables

Contexte : Gestion d'un inventaire de produits

```powershell
# Déclaration d'un tableau de produits
$Produits = @(
    @{ID = 1; Nom = "Ordinateur"; Prix = 800}
    @{ID = 2; Nom = "Clavier"; Prix = 50}
    @{ID = 3; Nom = "Souris"; Prix = 25}
)

# Afficher tous les produits
foreach ($Produit in $Produits) {
    Write-Output "ID: $($Produit.ID), Nom: $($Produit.Nom), Prix: $($Produit.Prix)€"
}

# Ajouter un produit
$NouveauProduit = @{ID = 4; Nom = "Écran"; Prix = 200}
$Produits += $NouveauProduit

# Filtrer les produits à plus de 100 €
$Produits | Where-Object { $_.Prix -gt 100 }
```

Variables et structures de données - Contrôle de flux

Variables et structures de données

Résumé des Concepts

Concept	Description
Variables	Stockent des données avec `$` comme préfixe.
Types de données	Incluent chaînes, nombres, booléens, etc.
Tableaux	Collections ordonnées d'éléments.
Hash Tables	Collections clé-valeur.

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

1. Structures Conditionnelles

a. `if`

Le `if` permet d'exécuter des blocs de code en fonction d'une ou plusieurs conditions.

Syntaxe :

```
if (<condition>) {
    <instructions>
} elseif (<condition>) {
    <instructions>
} else {
    <instructions>
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

1. Structures Conditionnelles

a. `if`

Le `if` permet d'exécuter des blocs de code en fonction d'une ou plusieurs conditions.

Exemple :

```
$Nombre = 10

if ($Nombre -lt 0) {
    Write-Output "Nombre négatif"
} elseif ($Nombre -eq 0) {
    Write-Output "Nombre nul"
} else {
    Write-Output "Nombre positif"
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

1. Structures Conditionnelles

b. `switch`

Le `switch` permet d'évaluer une variable ou une expression contre plusieurs cas.

Syntaxe :

```
switch (<expression>) {
    <valeur1> { <instructions> }
    <valeur2> { <instructions> }
    Default { <instructions> }
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

1. Structures Conditionnelles

b. `switch`

Le `switch` permet d'évaluer une variable ou une expression contre plusieurs cas.

Exemple :

```
$Jour = "Mardi"

switch ($Jour) {
    "Lundi" { Write-Output "Début de la semaine" }
    "Samedi" { Write-Output "Weekend !" }
    "Dimanche" { Write-Output "Weekend !" }
    Default { Write-Output "Jour ordinaire" }
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

2. Boucles

a. `for`

La boucle `for` est utilisée pour exécuter un bloc de code un nombre défini de fois.

Syntaxe :

```
for (<initialisation>; <condition>; <incrément>) {
    <instructions>
}
```

Exemple :

```
for ($i = 1; $i -le 5; $i++) {
    Write-Output "Itération $i"
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

2. Boucles

b. `foreach`

La boucle `foreach` permet d'itérer sur chaque élément d'une collection.

Syntaxe :

```
foreach ($element in <collection>) {
    <instructions>
}
```

Exemple :

```
$Liste = @(10, 20, 30)

foreach ($Valeur in $Liste) {
    Write-Output "Valeur : $Valeur"
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs en PowerShell

2. Boucles

c. `while`

La boucle `while` exécute un bloc de code tant qu'une condition est vraie.

Syntaxe :

```
while (<condition>) {
    <instructions>
}
```

Exemple :

```
$Compteur = 1
while ($Compteur -le 5) {
    Write-Output "Compteur : $Compteur"
    $Compteur++
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

2. Boucles

d. `do-until`

La boucle `do-until` exécute un bloc de code au moins une fois, puis continue tant qu'une condition est fausse.

Syntaxe :

```
do {
    <instructions>
} until (<condition>)
```

Exemple :

```
$Compteur = 1
do {
    Write-Output "Compteur : $Compteur"
    $Compteur++
} until ($Compteur -gt 5)
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

3. Utilisation des Filtres (`Where-Object`)

Description :

`Where-Object` filtre une collection d'objets selon une condition définie.

Syntaxe :

```
<collection> | Where-Object { <condition> }
```

Exemples :

1. **Filtrer les fichiers de plus de 1 Mo dans un dossier :**

```
Get-ChildItem -Path C:\Temp | Where-Object { $_.Length -gt 1MB }
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

3. Utilisation des Filtres (`Where-Object`)

Exemples :

2. **Afficher les processus consommant plus de 10 % de CPU :**

```
Get-Process | Where-Object { $_.CPU -gt 10 }
```

3. **Lister les services démarrés :**

```
Get-Service | Where-Object { $_.Status -eq "Running" }
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

4. Utilisation des Sélecteurs (`Select-Object`)

Description :

`Select-Object` sélectionne des propriétés spécifiques d'une collection d'objets ou limite les résultats.

Syntaxe :

```
<collection> | Select-Object [-Property <Propriétés>] [-First <Nombre>] [-Last <Nombre>]
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

4. Utilisation des Sélecteurs (`Select-Object`)

Exemples :

1. **Sélectionner des propriétés spécifiques :**

```
Get-Process | Select-Object Name, CPU
```

2. **Limiter les résultats aux 5 premiers éléments :**

```
Get-ChildItem -Path C:\Temp | Select-Object -First 5
```

3. **Récupérer les 3 derniers fichiers modifiés :**

```
Get-ChildItem -Path C:\Temp | Sort-Object LastWriteTime -Descending | Select-Object -Last 3
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

4. Utilisation des Sélecteurs (`Select-Object`)

Exemple Complet

Contexte : Gestion des fichiers dans un répertoire

```
# Parcourir les fichiers d'un répertoire
$Fichiers = Get-ChildItem -Path C:\Temp

# Filtrer les fichiers de plus de 1 Mo
$FichiersGrands = $Fichiers | Where-Object { $_.Length -gt 1MB }

# Trier les fichiers par taille décroissante et sélectionner les 3 premiers
$TopFichiers = $FichiersGrands | Sort-Object -Property Length -Descending | Select-Object -First 3

# Afficher les fichiers sélectionnés
foreach ($Fichier in $TopFichiers) {
    Write-Output "Nom : $($Fichier.Name), Taille : $($Fichier.Length)"
}
```

Variables et structures de données - Contrôle de flux

Structures Conditionnelles, Boucles et Filtres/Sélecteurs

4. Utilisation des Sélecteurs (`Select-Object`)

Résumé des Cmdlets

Concept	Cmdlet ou Structure	Description
Conditionnelles	`if`, `switch`	Exécution conditionnelle des blocs de code.
Boucles	`for`, `foreach`, `while`, `do-until`	Répétition de blocs de code selon des conditions.
Filtrage	`Where-Object`	Filtre les objets en fonction de conditions.
Sélection	`Select-Object`	Sélectionne des propriétés ou limite les résultats.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Gestion des systèmes et des services Windows

Gestion des systèmes et des services Windows

Gestion des Processus Windows

- PowerShell offre des cmdlets puissantes pour gérer les processus sur un système Windows, comme les surveiller, les terminer ou interagir avec eux.

Gestion des systèmes et des services Windows

Gestion des Processus Windows

1. Gestion des Processus avec `Get-Process` et Autres Cmdlets

a. `Get-Process`

- Liste tous les processus en cours d'exécution.
- Affiche des détails tels que le nom, l'ID (PID), l'utilisation CPU, la mémoire, etc.

Syntaxe :

```
Get-Process [-Name <Nom>] [-Id <ID>] [-ComputerName <NomMachine>]
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

1. Gestion des Processus avec `Get-Process` et Autres Cmdlets

Exemples :

1. **Lister tous les processus :**

```
Get-Process
```

2. **Filtrer par nom de processus :**

```
Get-Process -Name notepad
```

3. **Filtrer par ID de processus :**

```
Get-Process -Id 1234
```

4. **Obtenir des processus sur une machine distante :**

```
Get-Process -ComputerName "Serveur01"
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

1. Gestion des Processus avec `Get-Process` et Autres Cmdlets

b. `Stop-Process`

- Arrête un processus en fonction de son nom ou de son ID.

Syntaxe :

```
Stop-Process [-Name <Nom>] [-Id <ID>] [-Force]
```

Exemples :

1. **Arrêter un processus par son nom :**

   ```
   Stop-Process -Name notepad
   ```

2. **Arrêter un processus par son ID :**

   ```
   Stop-Process -Id 1234
   ```

3. **Forcer l'arrêt d'un processus :**

   ```
   Stop-Process -Name notepad -Force
   ```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

1. Gestion des Processus avec `Get-Process` et Autres Cmdlets

c. `Start-Process`

- Permet de lancer un nouveau processus.

Syntaxe :

```
Start-Process [-FilePath <Chemin>] [-ArgumentList <Arguments>] [-Wait]
```

Exemples :

1. **Ouvrir une application (ex : Bloc-notes) :**

   ```
   Start-Process -FilePath "notepad.exe"
   ```

2. **Ouvrir un fichier avec une application par défaut :**

   ```
   Start-Process -FilePath "C:\Temp\document.txt"
   ```

3. **Exécuter une commande en attendant sa fin :**

   ```
   Start-Process -FilePath "cmd.exe" -ArgumentList "/C ping google.com" -Wait
   ```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

1. Gestion des Processus avec `Get-Process` et Autres Cmdlets

d. Surveillance des Processus

1. **Lister les processus triés par utilisation CPU décroissante :**

```
Get-Process | Sort-Object CPU -Descending
```

2. **Afficher les 5 processus consommant le plus de mémoire :**

```
Get-Process | Sort-Object WorkingSet -Descending | Select-Object -First 5
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

Les services Windows peuvent être surveillés et contrôlés via PowerShell à l'aide de cmdlets comme `Get-Service`, `Start-Service`, et `Stop-Service`.

a. `Get-Service`

- Liste les services installés sur la machine, qu'ils soient en cours d'exécution ou non.

Syntaxe :

```
Get-Service [-Name <Nom>] [-DisplayName <NomAffichage>] [-ComputerName <NomMachine>]
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

Exemples :

1. **Lister tous les services :**

```
Get-Service
```

2. **Afficher les services en cours d'exécution uniquement :**

```
Get-Service | Where-Object { $_.Status -eq "Running" }
```

3. **Filtrer par nom de service :**

```
Get-Service -Name W32Time
```

4. **Lister les services d'une machine distante :**

```
Get-Service -ComputerName "Serveur01"
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

b. `Start-Service`

- Démarre un service arrêté.

Syntaxe :

```
Start-Service -Name <Nom> [-Force]
```

Exemples :

1. **Démarrer un service spécifique :**

   ```
   Start-Service -Name W32Time
   ```

2. **Forcer le démarrage d'un service :**

   ```
   Start-Service -Name Spooler -Force
   ```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

c. `Stop-Service`

- Arrête un service en cours d'exécution.

Syntaxe :

```
Stop-Service -Name <Nom> [-Force]
```

Exemples :

1. **Arrêter un service :**

   ```
   Stop-Service -Name W32Time
   ```

2. **Forcer l'arrêt d'un service :**

   ```
   Stop-Service -Name Spooler -Force
   ```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

d. `Restart-Service`

- Redémarre un service.

Syntaxe :

```
Restart-Service -Name <Nom> [-Force]
```

Exemples :

1. **Redémarrer un service :**

   ```
   Restart-Service -Name W32Time
   ```

2. **Forcer le redémarrage :**

   ```
   Restart-Service -Name Spooler -Force
   ```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

2. Gestion des Services Windows

Exemple Complet

1. Gestion des Processus

```
# Lister les processus avec plus de 10% de CPU
Get-Process | Where-Object { $_.CPU -gt 10 }

# Terminer tous les processus nommés "notepad"
Get-Process -Name notepad | Stop-Process
```

2. Gestion des Services

```
# Afficher les services en cours d'exécution
Get-Service | Where-Object { $_.Status -eq "Running" }

# Démarrer le service de temps Windows
Start-Service -Name W32Time

# Arrêter le service d'impression
Stop-Service -Name Spooler
```

Gestion des systèmes et des services Windows

Gestion des Processus Windows

Résumé des Cmdlets

Cmdlet	Description
`Get-Process`	Liste les processus en cours.
`Start-Process`	Démarre une nouvelle application ou commande.
`Stop-Process`	Termine un processus spécifique.
`Get-Service`	Liste les services installés sur le système.
`Start-Service`	Démarre un service arrêté.
`Stop-Service`	Arrête un service en cours d'exécution.
`Restart-Service`	Redémarre un service.

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

1. Création et Gestion des Comptes Utilisateurs Locaux

Pour gérer les utilisateurs locaux, PowerShell utilise les cmdlets du module `Microsoft.PowerShell.LocalAccounts`. Ce module est disponible à partir de Windows 10 et Windows Server 2016.

a. Création d'un utilisateur local

Cmdlet : `New-LocalUser`

- Permet de créer un utilisateur local.

Syntaxe :

```
New-LocalUser -Name <NomUtilisateur> -Password <MotDePasse> [-FullName <NomComplet>] [-Description <Description>]
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

1. Création et Gestion des Comptes Utilisateurs Locaux

a. Création d'un utilisateur local

Exemple :

```
$Password = ConvertTo-SecureString "P@ssw0rd" -AsPlainText -Force
New-LocalUser -Name "UtilisateurTest" -Password $Password -FullName "Utilisateur de Test" -Description "Compte utilisateur de test"
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

1. Création et Gestion des Comptes Utilisateurs Locaux

b. Modification d'un utilisateur local

Cmdlet : `Set-LocalUser`

- Permet de modifier les propriétés d'un utilisateur local.

Syntaxe :

```
Set-LocalUser -Name <NomUtilisateur> [-Password <MotDePasse>] [-Description <Description>]
```

Exemple :

```
Set-LocalUser -Name "UtilisateurTest" -Description "Compte modifié"
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

1. Création et Gestion des Comptes Utilisateurs Locaux

c. Suppression d'un utilisateur local

Cmdlet : `Remove-LocalUser`

- Permet de supprimer un utilisateur local.

Syntaxe :

```
Remove-LocalUser -Name <NomUtilisateur>
```

Exemple :

```
Remove-LocalUser -Name "UtilisateurTest"
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

1. Création et Gestion des Comptes Utilisateurs Locaux

d. Lister les utilisateurs locaux

Cmdlet : `Get-LocalUser`

- Liste les utilisateurs locaux sur la machine.

Syntaxe :

```
Get-LocalUser
```

Exemple :

```
Get-LocalUser | Format-Table Name, FullName, Enabled
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

2. Modification des Permissions avec PowerShell

Pour gérer les permissions sur des fichiers, des dossiers ou d'autres ressources, PowerShell utilise les cmdlets associées au module `NTFS` ou à l'API .NET via les classes ACL.

a. Récupérer les permissions d'un fichier ou dossier

Cmdlet : `Get-Acl`

- Permet de récupérer les informations ACL d'un fichier ou d'un dossier.

Syntaxe :

```
Get-Acl -Path <Chemin>
```

Exemple :

```
Get-Acl -Path "C:\DossierTest"
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

2. Modification des Permissions avec PowerShell

b. Modifier les permissions

Cmdlet : `Set-Acl`

- Applique une liste de contrôle d'accès (ACL) à un fichier ou dossier.

Syntaxe :

```
$Acl = Get-Acl -Path <Chemin>
$Acl.SetAccessRule(<RègleAccès>)
Set-Acl -Path <Chemin> -AclObject $Acl
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

2. Modification des Permissions avec PowerShell

b. Modifier les permissions

Exemple :

1. **Créer une règle d'accès :**

```
$Acl = Get-Acl -Path "C:\DossierTest"
$Règle = New-Object System.Security.AccessControl.FileSystemAccessRule("UtilisateurTest", "FullControl", "Allow")
$Acl.SetAccessRule($Règle)
Set-Acl -Path "C:\DossierTest" -AclObject $Acl
```

2. **Supprimer une règle d'accès :**

```
$Règle = New-Object System.Security.AccessControl.FileSystemAccessRule("UtilisateurTest", "FullControl", "Allow")
$Acl.RemoveAccessRule($Règle)
Set-Acl -Path "C:\DossierTest" -AclObject $Acl
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

3. Introduction à Active Directory

Active Directory (AD) est un service annuaire utilisé pour gérer les utilisateurs, groupes et ressources dans un environnement réseau. PowerShell offre des cmdlets via le module **Active Directory** (module AD).

a. Installation du module Active Directory

1. **Sur un contrôleur de domaine Windows Server :**

```
Install-WindowsFeature -Name RSAT-AD-PowerShell
```

2. **Sur un poste Windows 10/11 ou Windows Server :**
   - Installer les outils d'administration distants (RSAT) via les fonctionnalités facultatives :

```
Add-WindowsCapability -Online -Name Rsat.ActiveDirectory.DS-LDS.Tools~~~~0.0.1.0
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

3. Introduction à Active Directory

b. Cmdlets pour gérer les objets Active Directory

Cmdlet	Description
`Get-ADUser`	Récupère des informations sur un utilisateur.
`New-ADUser`	Crée un nouvel utilisateur dans AD.
`Set-ADUser`	Modifie les propriétés d'un utilisateur AD.
`Remove-ADUser`	Supprime un utilisateur de l'Active Directory.
`Get-ADGroup`	Récupère des informations sur un groupe.
`Add-ADGroupMember`	Ajoute un utilisateur ou un objet à un groupe.
`Get-ADComputer`	Récupère des informations sur un ordinateur AD.

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

3. Introduction à Active Directory

c. Exemples Pratiques

1. Lister tous les utilisateurs d'un domaine :

```
Get-ADUser -Filter * | Select-Object Name, SamAccountName, Enabled
```

2. Créer un nouvel utilisateur :

```
New-ADUser -Name "UtilisateurAD" -SamAccountName "UtilisateurAD" -UserPrincipalName "utilisateur@domaine.local" -Path "OU=Utilisateurs,DC=domaine,DC=local" -AccountPassword (ConvertTo-SecureString "P@ssw0rd!" -AsPlainText -Force) -Enabled $true
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

3. Introduction à Active Directory

c. Exemples Pratiques

3. Ajouter un utilisateur à un groupe :

```
Add-ADGroupMember -Identity "Administrateurs" -Members "UtilisateurAD"
```

4. Modifier les informations d'un utilisateur :

```
Set-ADUser -Identity "UtilisateurAD" -Title "Développeur" -Department "Informatique"
```

5. Supprimer un utilisateur :

```
Remove-ADUser -Identity "UtilisateurAD"
```

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

Résumé

Cmdlet	Description
`New-LocalUser`	Crée un utilisateur local.
`Set-LocalUser`	Modifie un utilisateur local.
`Remove-LocalUser`	Supprime un utilisateur local.
`Get-LocalUser`	Liste les utilisateurs locaux.

Cmdlet	Description
`Get-Acl`	Récupère les permissions d’un fichier/dossier.
`Set-Acl`	Modifie les permissions d’un fichier/dossier.

Gestion des systèmes et des services Windows

Gestion des Comptes Utilisateurs et Permissions

Résumé

Active Directory :

Cmdlet	Description
`Get-ADUser`	Liste ou récupère un utilisateur AD.
`New-ADUser`	Crée un nouvel utilisateur dans AD.
`Set-ADUser`	Modifie les propriétés d'un utilisateur AD.
`Add-ADGroupMember`	Ajoute un utilisateur à un groupe AD.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Automatisation avancée

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

PowerShell permet de créer, configurer et gérer des tâches planifiées à l'aide des cmdlets associées au module **ScheduledTasks**.

a. Création d'une tâche planifiée

Cmdlet : `New-ScheduledTask` et `Register-ScheduledTask`

- `New-ScheduledTask` définit une tâche planifiée.
- `Register-ScheduledTask` enregistre la tâche pour qu'elle s'exécute automatiquement.

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

PowerShell permet de créer, configurer et gérer des tâches planifiées à l'aide des cmdlets associées au module **ScheduledTasks**.

a. Création d'une tâche planifiée

Syntaxe :

```
$Trigger = New-ScheduledTaskTrigger -Daily -At "08:00AM"
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File C:\Scripts\MonScript.ps1"
Register-ScheduledTask -TaskName "MaTache" -Trigger $Trigger -Action $Action -Description "Exemple de tâche planifiée"
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

a. Création d'une tâche planifiée

Exemple complet :

```
# Définir le déclencheur (tâche quotidienne à 8h)
$Trigger = New-ScheduledTaskTrigger -Daily -At "08:00AM"

# Définir l'action (exécution d'un script PowerShell)
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File C:\Scripts\Backup.ps1"

# Enregistrer la tâche planifiée
Register-ScheduledTask -TaskName "BackupJournalier" -Trigger $Trigger -Action $Action -Description "Tâche de sauvegarde quotidienne"
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

b. Modifier une tâche existante

Cmdlet : `Set-ScheduledTask`

- Permet de modifier les propriétés d'une tâche existante.

Exemple :

```
Set-ScheduledTask -TaskName "BackupJournalier" -Trigger (New-ScheduledTaskTrigger -Weekly -DaysOfWeek Monday -At "06:00AM")
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

c. Lister les tâches planifiées

Cmdlet : `Get-ScheduledTask`

- Liste toutes les tâches planifiées.

Exemple :

```
Get-ScheduledTask | Where-Object { $_.TaskName -like "*Backup*" }
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

1. Planification des Tâches avec PowerShell

d. Supprimer une tâche planifiée

Cmdlet : `Unregister-ScheduledTask`

- Supprime une tâche planifiée.

Exemple :

```
Unregister-ScheduledTask -TaskName "BackupJournalier" -Confirm:$false
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

2. Lecture et Analyse des Journaux d'Événements Windows

Les journaux d'événements Windows sont une source précieuse pour diagnostiquer des problèmes ou surveiller des systèmes.

a. Cmdlet : `Get-EventLog`

- Utilisée pour lire les journaux d'événements traditionnels (applications, sécurité, système).

Syntaxe :

```
Get-EventLog -LogName <NomJournal> [-Newest <Nombre>] [-EntryType <Type>]
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

2. Lecture et Analyse des Journaux d'Événements Windows

a. Cmdlet : `Get-EventLog`

Exemple :

1. **Lister les 10 dernières entrées du journal système :**

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

2. **Filtrer les erreurs dans le journal application :**

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

3. **Exporter le journal système dans un fichier CSV :**

```
Get-EventLog -LogName System -Newest 50 | Export-Csv -Path C:\Logs\SystemEvents.csv -NoTypeInformation
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

2. Lecture et Analyse des Journaux d'Événements Windows

b. Cmdlet : `Get-WinEvent`

- Recommandée pour lire les journaux d'événements modernes (plus flexible que `Get-EventLog`).

Syntaxe :

```
Get-WinEvent -LogName <NomJournal> [-MaxEvents <Nombre>] [-FilterHashTable <Filtres>]
```

Exemple :

1. **Lister les événements récents du journal Sécurité :**

```
Get-WinEvent -LogName Security -MaxEvents 10
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

2. Lecture et Analyse des Journaux d'Événements Windows

b. Cmdlet : `Get-WinEvent`

2. **Filtrer par ID d'événement :**

```
Get-WinEvent -LogName Application | Where-Object { $_.Id -eq 1000 }
```

3. **Filtrer par date :**

```
Get-WinEvent -LogName System -FilterHashtable @{StartTime="2024-12-01"; EndTime="2024-12-31"}
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

3. Génération de Rapports Automatisés

PowerShell permet de générer des rapports dans des formats comme CSV ou HTML pour une analyse visuelle.

a. Génération de rapports CSV

Cmdlet : `Export-Csv`

- Convertit une sortie en fichier CSV.

Exemple :

1. **Exporter la liste des services en cours d'exécution :**

```
Get-Service | Where-Object { $_.Status -eq "Running" } | Export-Csv -Path C:\Rapports\ServicesRunning.csv -NoTypeInformation
```

2. **Exporter les 10 processus les plus gourmands en mémoire :**

```
Get-Process | Sort-Object -Property WorkingSet -Descending | Select-Object -First 10 | Export-Csv -Path C:\Rapports\TopMemoryProcesses.csv -NoTypeInformation
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

3. Génération de Rapports Automatisés

b. Génération de rapports HTML

Cmdlet : `ConvertTo-Html`

- Convertit une sortie en tableau HTML.

Exemple :

1. **Créer un rapport HTML des 5 derniers fichiers modifiés :**

```
Get-ChildItem -Path C:\Dossiers -Recurse | Sort-Object -Property LastWriteTime -Descending | Select-Object -First 5 | ConvertTo-Html -Property Name, LastWriteTime, Length -Title "Derniers Fichiers Modifiés" | Out-File -FilePath C:\Rapports\FichiersRecents.html
```

2. **Rapport des événements système :**

```
Get-EventLog -LogName System -Newest 20 | ConvertTo-Html -Property EntryType, Source, TimeGenerated, Message -Title "Rapport des Événements Système" | Out-File -FilePath C:\Rapports\SystemEvents.html
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

3. Génération de Rapports Automatisés

c. Génération automatique avec un script

Exemple de script pour un rapport journalier :

```
# Chemin du rapport
$RapportHTML = "C:\Rapports\RapportSysteme.html"

# Récupérer les événements récents
$Evenements = Get-EventLog -LogName System -Newest 50

# Générer un rapport HTML
$Evenements | ConvertTo-Html -Property EntryType, Source, TimeGenerated, Message -Title "Rapport Système Journalier" | Out-File -FilePath $RapportHTML

Write-Output "Rapport généré : $RapportHTML"
```

Automatisation avancée

Planification des Tâches, Analyse des Journaux et Génération de Rapports

Tâche	Cmdlet	Description
Planification des tâches	`New-ScheduledTask`, `Register-ScheduledTask`, `Set-ScheduledTask`	Création et modification des tâches planifiées.
Journaux d'événements	`Get-EventLog`, `Get-WinEvent`	Lecture et filtrage des journaux d'événements.
Génération de rapports (CSV)	`Export-Csv`	Export des données structurées en fichier CSV.
Génération de rapports (HTML)	`ConvertTo-Html`	Conversion des données en tableaux HTML.

Automatisation avancée

Création de Scripts PowerShell

1. Création de Scripts PowerShell : Structure et Bonnes Pratiques

Structure d'un script PowerShell

1. **Introduction et informations** :
   - Utiliser des commentaires pour expliquer le script.
   - Ajouter des métadonnées sur le script.

```
# Nom : ScriptExemple.ps1
# Description : Automatisation des tâches système.
# Auteur : Votre Nom
# Date : 31-12-2024
```

2. **Importation des modules nécessaires** :

```
Import-Module ActiveDirectory
```

Automatisation avancée

Création de Scripts PowerShell

1. Création de Scripts PowerShell : Structure et Bonnes Pratiques

3. **Déclaration des paramètres** :
   - Utiliser la section `Param` pour des entrées dynamiques.

```
param(
    [string]$NomUtilisateur,
    [int]$NombreIterations
)
```

4. **Logique principale** :
   - Ajouter des blocs logiques bien séparés pour les différentes tâches.
5. **Gestion des erreurs** :
   - Intégrer des blocs `Try-Catch-Finally` pour capturer les erreurs.

Automatisation avancée

Création de Scripts PowerShell

1. Création de Scripts PowerShell : Structure et Bonnes Pratiques

Bonnes pratiques

1. **Nommer les scripts de manière descriptive** :
   - Exemple : `CréerUtilisateurAD.ps1`.
2. **Utiliser des commentaires** :

```
# Cette section configure les permissions pour un dossier.
```

3. **Éviter le code en dur** :
   - Utiliser des paramètres dynamiques ou des fichiers de configuration.
4. **Tester les scripts en environnement contrôlé**.
5. **Utiliser la sortie structurée (CSV, JSON)** pour les données générées.

Automatisation avancée

Création de Scripts PowerShell

1. Création de Scripts PowerShell : Structure et Bonnes Pratiques

Exemple de script simple : Gestion des utilisateurs locaux

```
# Script : AjouterUtilisateur.ps1
# Description : Crée un utilisateur local avec des paramètres personnalisés.

param(
    [string]$NomUtilisateur,
    [string]$MotDePasse
)

try {
    # Convertir le mot de passe en chaîne sécurisée
    $Password = ConvertTo-SecureString $MotDePasse -AsPlainText -Force

    # Créer l'utilisateur
    New-LocalUser -Name $NomUtilisateur -Password $Password -FullName "Utilisateur $NomUtilisateur" -Description "Compte créé via script"

    Write-Output "Utilisateur $NomUtilisateur créé avec succès."
} catch {
    Write-Error "Erreur lors de la création de l'utilisateur : $_"
}
```

Automatisation avancée

Création de Scripts PowerShell

2. Introduction aux Fonctions PowerShell

Déclaration d'une fonction

- Les fonctions encapsulent des blocs de code réutilisables.

Syntaxe :

```
function <NomFonction> {
    param(
        [Type]$Param1,
        [Type]$Param2
    )
    <Instructions>
    return <Valeur>
}
```

Exemples pratiques

a. Fonction sans paramètre

```
function DireBonjour {
    Write-Output "Bonjour, PowerShell !"
}
DireBonjour
```

Automatisation avancée

Création de Scripts PowerShell

2. Introduction aux Fonctions PowerShell

b. Fonction avec paramètres

```
function Addition {
    param(
        [int]$Nombre1,
        [int]$Nombre2
    )
    return $Nombre1 + $Nombre2
}
$Resultat = Addition -Nombre1 10 -Nombre2 5
Write-Output "Résultat : $Resultat"
```

c. Fonction avec valeur par défaut

```
function DireBonjour {
    param(
        [string]$Nom = "Utilisateur"
    )
    Write-Output "Bonjour, $Nom !"
}
DireBonjour
DireBonjour -Nom "Alice"
```

Automatisation avancée

Création de Scripts PowerShell

2. Introduction aux Fonctions PowerShell

d. Fonction réutilisable pour gérer des services

```
function RedemarrerService {
    param(
        [string]$NomService
    )
    try {
        Restart-Service -Name $NomService -Force
        Write-Output "Le service $NomService a été redémarré."
    } catch {
        Write-Error "Erreur lors du redémarrage du service $NomService : $_"
    }
}
RedemarrerService -NomService "Spooler"
```

Automatisation avancée

Création de Scripts PowerShell

3. Gestion des Erreurs avec `Try-Catch-Finally`

PowerShell permet de gérer les erreurs avec les blocs `Try-Catch-Finally`.

Structure de base

```
try {
    # Bloc où une erreur peut survenir
    <Instructions>
} catch {
    # Bloc exécuté en cas d'erreur
    Write-Error "Une erreur est survenue : $_"
} finally {
    # Bloc exécuté dans tous les cas (optionnel)
    Write-Output "Opération terminée."
}
```

Automatisation avancée

Création de Scripts PowerShell

3. Gestion des Erreurs avec `Try-Catch-Finally`

Exemples pratiques

a. Gestion d'une erreur simple

```
try {
    # Suppression d'un fichier inexistant
    Remove-Item -Path "C:\Inexistant.txt" -ErrorAction Stop
} catch {
    Write-Error "Erreur : $_"
} finally {
    Write-Output "Bloc Finally exécuté."
}
```

Automatisation avancée

Création de Scripts PowerShell

3. Gestion des Erreurs avec `Try-Catch-Finally`

Exemples pratiques

b. Valider un chemin avant d'accéder à un fichier

```
function LireFichier {
    param(
        [string]$CheminFichier
    )
    try {
        if (-Not (Test-Path $CheminFichier)) {
            throw "Le fichier $CheminFichier n'existe pas."
        }

        $Contenu = Get-Content -Path $CheminFichier
        Write-Output "Contenu du fichier :"
        Write-Output $Contenu
    } catch {
        Write-Error "Erreur : $_"
    }
}

LireFichier -CheminFichier "C:\Inexistant.txt"
```

Automatisation avancée

Création de Scripts PowerShell

3. Gestion des Erreurs avec `Try-Catch-Finally`

Exemples pratiques

c. Blocs imbriqués pour des scénarios complexes

```
try {
    Write-Output "Début de l'opération."
    # Exemple de tentative d'exécution
    try {
        $Resultat = 10 / 0  # Erreur intentionnelle
    } catch {
        throw "Erreur dans l'opération interne : $_"
    }
} catch {
    Write-Error "Une erreur majeure est survenue : $_"
} finally {
    Write-Output "Nettoyage en cours..."
}
```

Automatisation avancée

Création de Scripts PowerShell

Résumé des Concepts

Concept	Description
Scripts	Code réutilisable, structuré en fichiers `.ps1`.
Fonctions	Blocs de code réutilisables, avec ou sans paramètres.
Try-Catch-Finally	Gestion robuste des erreurs dans PowerShell.

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

PowerShell permet de créer, configurer et gérer des tâches planifiées à l'aide des cmdlets associées au module **ScheduledTasks**.

a. Création d'une tâche planifiée

Cmdlet : `New-ScheduledTask` et `Register-ScheduledTask`

- `New-ScheduledTask` définit une tâche planifiée.
- `Register-ScheduledTask` enregistre la tâche pour qu'elle s'exécute automatiquement.

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

a. Création d'une tâche planifiée

Syntaxe :

```
$Trigger = New-ScheduledTaskTrigger -Daily -At "08:00AM"
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File C:\Scripts\MonScript.ps1"
Register-ScheduledTask -TaskName "MaTache" -Trigger $Trigger -Action $Action -Description "Exemple de tâche planifiée"
```

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

a. Création d'une tâche planifiée

Exemple complet :

```
# Définir le déclencheur (tâche quotidienne à 8h)
$Trigger = New-ScheduledTaskTrigger -Daily -At "08:00AM"

# Définir l'action (exécution d'un script PowerShell)
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File C:\Scripts\Backup.ps1"

# Enregistrer la tâche planifiée
Register-ScheduledTask -TaskName "BackupJournalier" -Trigger $Trigger -Action $Action -Description "Tâche de sauvegarde quotidienne"
```

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

b. Modifier une tâche existante

Cmdlet : `Set-ScheduledTask`

- Permet de modifier les propriétés d'une tâche existante.

Exemple :

```
Set-ScheduledTask -TaskName "BackupJournalier" -Trigger (New-ScheduledTaskTrigger -Weekly -DaysOfWeek Monday -At "06:00AM")
```

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

c. Lister les tâches planifiées

Cmdlet : `Get-ScheduledTask`

- Liste toutes les tâches planifiées.

Exemple :

```
Get-ScheduledTask | Where-Object { $_.TaskName -like "*Backup*" }
```

Automatisation avancée

Tâches planifiées et journaux

1. Planification des Tâches avec PowerShell

d. Supprimer une tâche planifiée

Cmdlet : `Unregister-ScheduledTask`

- Supprime une tâche planifiée.

Exemple :

```
Unregister-ScheduledTask -TaskName "BackupJournalier" -Confirm:$false
```

Automatisation avancée

Tâches planifiées et journaux

2. Lecture et Analyse des Journaux d'Événements Windows

Les journaux d'événements Windows sont une source précieuse pour diagnostiquer des problèmes ou surveiller des systèmes.

a. Cmdlet : `Get-EventLog`

- Utilisée pour lire les journaux d'événements traditionnels (applications, sécurité, système).

Syntaxe :

```
Get-EventLog -LogName <NomJournal> [-Newest <Nombre>] [-EntryType <Type>]
```

Exemple :

1. **Lister les 10 dernières entrées du journal système :**

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

Automatisation avancée

Tâches planifiées et journaux

2. Lecture et Analyse des Journaux d'Événements Windows

Les journaux d'événements Windows sont une source précieuse pour diagnostiquer des problèmes ou surveiller des systèmes.

a. Cmdlet : `Get-EventLog`

Exemple :

2. **Filtrer les erreurs dans le journal application :**

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

3. **Exporter le journal système dans un fichier CSV :**

```
Get-EventLog -LogName System -Newest 50 | Export-Csv -Path C:\Logs\SystemEvents.csv -NoTypeInformation
```

Automatisation avancée

Tâches planifiées et journaux

2. Lecture et Analyse des Journaux d'Événements Windows

b. Cmdlet : `Get-WinEvent`

- Recommandée pour lire les journaux d'événements modernes (plus flexible que `Get-EventLog`).

Syntaxe :

```
Get-WinEvent -LogName <NomJournal> [-MaxEvents <Nombre>] [-FilterHashTable <Filtres>]
```

Exemple :

1. **Lister les événements récents du journal Sécurité :**

   ```
   Get-WinEvent -LogName Security -MaxEvents 10
   ```

2. **Filtrer par ID d'événement :**

   ```
   Get-WinEvent -LogName Application | Where-Object { $_.Id -eq 1000 }
   ```

Automatisation avancée

Tâches planifiées et journaux

2. Lecture et Analyse des Journaux d'Événements Windows

b. Cmdlet : `Get-WinEvent`

Exemple :

3. **Filtrer par date :**

```
Get-WinEvent -LogName System -FilterHashtable @{StartTime="2024-12-01"; EndTime="2024-12-31"}
```

Automatisation avancée

Tâches planifiées et journaux

3. Génération de Rapports Automatisés

PowerShell permet de générer des rapports dans des formats comme CSV ou HTML pour une analyse visuelle.

a. Génération de rapports CSV

Cmdlet : `Export-Csv`

- Convertit une sortie en fichier CSV.

Exemple :

1. **Exporter la liste des services en cours d'exécution :**

```
Get-Service | Where-Object { $_.Status -eq "Running" } | Export-Csv -Path C:\Rapports\ServicesRunning.csv -NoTypeInformation
```

2. **Exporter les 10 processus les plus gourmands en mémoire :**

```
Get-Process | Sort-Object -Property WorkingSet -Descending | Select-Object -First 10 | Export-Csv -Path C:\Rapports\TopMemoryProcesses.csv -NoTypeInformation
```

Automatisation avancée

Tâches planifiées et journaux

3. Génération de Rapports Automatisés

PowerShell permet de générer des rapports dans des formats comme CSV ou HTML pour une analyse visuelle.

b. Génération de rapports HTML

Cmdlet : `ConvertTo-Html`

- Convertit une sortie en tableau HTML.

Exemple :

1. **Créer un rapport HTML des 5 derniers fichiers modifiés :**

```
Get-ChildItem -Path C:\Dossiers -Recurse | Sort-Object -Property LastWriteTime -Descending | Select-Object -First 5 | ConvertTo-Html -Property Name, LastWriteTime, Length -Title "Derniers Fichiers Modifiés" | Out-File -FilePath C:\Rapports\FichiersRecents.html
```

2. **Rapport des événements système :**

```
Get-EventLog -LogName System -Newest 20 | ConvertTo-Html -Property EntryType, Source, TimeGenerated, Message -Title "Rapport des Événements Système" | Out-File -FilePath C:\Rapports\SystemEvents.html
```

Automatisation avancée

Tâches planifiées et journaux

3. Génération de Rapports Automatisés

PowerShell permet de générer des rapports dans des formats comme CSV ou HTML pour une analyse visuelle.

c. Génération automatique avec un script

Exemple de script pour un rapport journalier :

```
# Chemin du rapport
$RapportHTML = "C:\Rapports\RapportSysteme.html"

# Récupérer les événements récents
$Evenements = Get-EventLog -LogName System -Newest 50

# Générer un rapport HTML
$Evenements | ConvertTo-Html -Property EntryType, Source, TimeGenerated, Message -Title "Rapport Système Journalier" | Out-File -FilePath $RapportHTML

Write-Output "Rapport généré : $RapportHTML"
```

Automatisation avancée

Tâches planifiées et journaux

Tâche	Cmdlet	Description
Planification des tâches	`New-ScheduledTask`, `Register-ScheduledTask`, `Set-ScheduledTask`	Création et modification des tâches planifiées.
Journaux d'événements	`Get-EventLog`, `Get-WinEvent`	Lecture et filtrage des journaux d'événements.
Génération de rapports (CSV)	`Export-Csv`	Export des données structurées en fichier CSV.
Génération de rapports (HTML)	`ConvertTo-Html`	Conversion des données en tableaux HTML.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Réseaux et sécurité

Réseaux et sécurité

Gestion Réseau avec PowerShell

- PowerShell offre un ensemble riche de cmdlets pour gérer, surveiller et diagnostiquer des réseaux.
- Ces outils permettent de tester la connectivité, analyser les ports, gérer les adresses IP et configurer les interfaces réseau.

Réseaux et sécurité

Gestion Réseau avec PowerShell

1. Cmdlets Réseau de Base

a. Tester la connectivité : `Test-Connection`

- Permet de vérifier la connectivité réseau (similaire à `ping` en ligne de commande).

Syntaxe :

```
Test-Connection -ComputerName <NomOuIP> [-Count <NombreDePaquets>]
```

Exemples :

1. **Tester la connectivité vers une machine :**

   ```
   Test-Connection -ComputerName google.com
   ```

2. **Envoyer 5 paquets :**

   ```
   Test-Connection -ComputerName google.com -Count 5
   ```

3. **Tester plusieurs machines :**

   ```
   "google.com", "192.168.1.1" | ForEach-Object { Test-Connection -ComputerName $_ -Count 1 }
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

1. Cmdlets Réseau de Base

b. Obtenir des informations IP : `Get-NetIPAddress`

- Fournit des informations sur les adresses IP configurées sur la machine.

Syntaxe :

```
Get-NetIPAddress [-InterfaceAlias <NomInterface>] [-AddressFamily <IPv4|IPv6>]
```

Exemples :

1. **Lister toutes les adresses IP :**

   ```
   Get-NetIPAddress
   ```

2. **Afficher uniquement les adresses IPv4 :**

   ```
   Get-NetIPAddress -AddressFamily IPv4
   ```

3. **Obtenir les adresses IP pour une interface spécifique :**

   ```
   Get-NetIPAddress -InterfaceAlias "Ethernet"
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

1. Cmdlets Réseau de Base

c. Résolution DNS : `Resolve-DnsName`

- Effectue une résolution DNS pour un nom de domaine.

Syntaxe :

```
Resolve-DnsName -Name <NomDomaine>
```

Exemples :

1. **Résoudre un nom de domaine :**

   ```
   Resolve-DnsName -Name google.com
   ```

2. **Résolution DNS avec type d'enregistrement spécifique :**

   ```
   Resolve-DnsName -Name google.com -Type A
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

2. Analyse des Ports et des Connexions Réseau

a. Analyse des ports : `Test-NetConnection`

- Vérifie l'état des ports TCP sur une machine cible.

Syntaxe :

```
Test-NetConnection -ComputerName <NomOuIP> -Port <Port>
```

Exemples :

1. **Vérifier la connectivité TCP vers un serveur web :**

   ```
   Test-NetConnection -ComputerName google.com -Port 80
   ```

2. **Tester une connexion RDP (port 3389) :**

   ```
   Test-NetConnection -ComputerName 192.168.1.10 -Port 3389
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

2. Analyse des Ports et des Connexions Réseau

b. Afficher les connexions réseau actives : `Get-NetTCPConnection`

- Fournit des informations sur les connexions TCP actives.

Syntaxe :

```
Get-NetTCPConnection [-State <État>] [-LocalPort <Port>]
```

Exemples :

1. **Lister toutes les connexions actives :**

   ```
   Get-NetTCPConnection
   ```

2. **Filtrer les connexions par état :**

   ```
   Get-NetTCPConnection -State Established
   ```

3. **Afficher les connexions pour un port local spécifique :**

   ```
   Get-NetTCPConnection -LocalPort 80
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

3. Gestion des Interfaces Réseau

a. Lister les interfaces réseau : `Get-NetAdapter`

- Affiche les informations sur les interfaces réseau disponibles.

Syntaxe :

```
Get-NetAdapter [-Name <NomInterface>]
```

Exemples :

1. **Lister toutes les interfaces :**

   ```
   Get-NetAdapter
   ```

2. **Afficher une interface spécifique :**

   ```
   Get-NetAdapter -Name "Ethernet"
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

3. Gestion des Interfaces Réseau

b. Activer/Désactiver une interface réseau : `Enable-NetAdapter` / `Disable-NetAdapter`

- Permet d'activer ou de désactiver une interface réseau.

Syntaxe :

```
Enable-NetAdapter -Name <NomInterface>
Disable-NetAdapter -Name <NomInterface>
```

Exemples :

1. **Désactiver une interface réseau :**

   ```
   Disable-NetAdapter -Name "Wi-Fi"
   ```

2. **Activer une interface réseau :**

   ```
   Enable-NetAdapter -Name "Wi-Fi"
   ```

Réseaux et sécurité

Gestion Réseau avec PowerShell

3. Gestion des Interfaces Réseau

c. Configurer une adresse IP : `New-NetIPAddress`

- Assigne une adresse IP statique à une interface réseau.

Syntaxe :

```
New-NetIPAddress -InterfaceAlias <NomInterface> -IPAddress <AdresseIP> -PrefixLength <LongueurPréfixe> -DefaultGateway <Passerelle>
```

Exemple :

```
New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress 192.168.1.100 -PrefixLength 24 -DefaultGateway 192.168.1.1
```

Réseaux et sécurité

Gestion Réseau avec PowerShell

3. Gestion des Interfaces Réseau

d. Réinitialiser une interface réseau : `Restart-NetAdapter`

- Redémarre une interface réseau.

Syntaxe :

```
Restart-NetAdapter -Name <NomInterface>
```

Exemple :

```
Restart-NetAdapter -Name "Ethernet"
```

Réseaux et sécurité

Gestion Réseau avec PowerShell

Exemple Complet : Script de Diagnostic Réseau

```
# Script : Diagnostic Réseau
# Description : Teste la connectivité réseau, analyse les connexions et vérifie l'état des interfaces.

# Étape 1 : Tester la connectivité vers une cible
$Cible = "google.com"
Write-Output "Test de connectivité vers $Cible..."
Test-Connection -ComputerName $Cible -Count 4

# Étape 2 : Vérifier l'état des interfaces réseau
Write-Output "État des interfaces réseau :"
Get-NetAdapter | Select-Object Name, Status, LinkSpeed

# Étape 3 : Analyse des connexions réseau actives
Write-Output "Connexions TCP établies :"
Get-NetTCPConnection -State Established | Select-Object LocalAddress, LocalPort, RemoteAddress, RemotePort
```

Réseaux et sécurité

Cmdlet	Description
`Test-Connection`	Vérifie la connectivité réseau (ping).
`Get-NetIPAddress`	Liste les adresses IP configurées.
`Resolve-DnsName`	Effectue une résolution DNS.
`Test-NetConnection`	Vérifie l'état des ports TCP.
`Get-NetTCPConnection`	Affiche les connexions TCP actives.
`Get-NetAdapter`	Liste les interfaces réseau.
`Enable-NetAdapter`	Active une interface réseau.
`Disable-NetAdapter`	Désactive une interface réseau.
`New-NetIPAddress`	Configure une adresse IP statique.
`Restart-NetAdapter`	Redémarre une interface réseau.

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

1. Utilisation de PowerShell Remoting

PowerShell Remoting permet d'exécuter des commandes ou des scripts sur des machines distantes. Il utilise généralement le protocole **WS-Management** via WinRM.

a. Activer PowerShell Remoting

1. **Activer le Remoting sur la machine locale :**

```
Enable-PSRemoting -Force
```

2. **Configurer la confiance mutuelle pour les machines non-domainées (optionnel) :**
   - Ajouter les machines distantes dans la liste des hôtes de confiance :

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value "NomMachineOuIP"
```

3. **Vérifier l'état de WinRM :**

```
Get-Service WinRM
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

1. Utilisation de PowerShell Remoting

b. Exécuter des commandes à distance

1. **Exécuter une commande sur une machine distante :**

```
Invoke-Command -ComputerName "NomMachine" -ScriptBlock { Get-Service }
```

2. **Passer des informations d'identification :**

```
$Credentials = Get-Credential
Invoke-Command -ComputerName "NomMachine" -Credential $Credentials -ScriptBlock { Get-Process }
```

3. **Exécuter un script complet sur une machine distante :**

```
Invoke-Command -ComputerName "NomMachine" -FilePath "C:\Scripts\MonScript.ps1"
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

1. Utilisation de PowerShell Remoting

c. Sessions distantes persistantes

1. **Créer une session distante :**

```
$Session = New-PSSession -ComputerName "NomMachine"
```

2. **Exécuter des commandes dans une session :**

```
Invoke-Command -Session $Session -ScriptBlock { Get-Date }
```

3. **Fermer la session :**

```
Remove-PSSession -Session $Session
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

2. Sécurisation des Scripts PowerShell

a. Exécution restreinte

PowerShell utilise des stratégies d'exécution pour sécuriser les scripts.

1. **Vérifier la stratégie actuelle :**

```
Get-ExecutionPolicy
```

2. **Changer la stratégie :**
   - **Restreindre totalement :**

```
Set-ExecutionPolicy Restricted
```

   - **Autoriser les scripts signés :**

```
Set-ExecutionPolicy AllSigned
```

   - **Autoriser tous les scripts :**

```
Set-ExecutionPolicy Unrestricted
```

3. **Exécuter un script temporairement avec bypass :**

```
PowerShell.exe -ExecutionPolicy Bypass -File "C:\Scripts\MonScript.ps1"
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

2. Sécurisation des Scripts PowerShell

b. Signatures de script

1. **Créer ou obtenir un certificat de signature :**
   - Créer un certificat auto-signé :

```
New-SelfSignedCertificate -CertStoreLocation Cert:\CurrentUser\My -Subject "CN=ScriptSigning"
```

2. **Signer un script :**

```
Set-AuthenticodeSignature -FilePath "C:\Scripts\MonScript.ps1" -Certificate (Get-Item Cert:\CurrentUser\My\<Thumbprint>)
```

3. **Vérifier la signature :**

```
Get-AuthenticodeSignature -FilePath "C:\Scripts\MonScript.ps1"
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

3. Gestion des Certificats et du Chiffrement

a. Gestion des certificats

1. **Lister les certificats :**

```
Get-ChildItem Cert:\CurrentUser\My
```

2. **Exporter un certificat :**

```
Export-Certificate -Cert (Get-Item Cert:\CurrentUser\My\<Thumbprint>) -FilePath "C:\Certificats\MonCert.cer"
```

3. **Supprimer un certificat :**

```
Remove-Item Cert:\CurrentUser\My\<Thumbprint>
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

3. Gestion des Certificats et du Chiffrement

b. Chiffrement et déchiffrement

1. **Chiffrer un texte avec une clé sécurisée :**

```
$Texte = "Message secret"
$Clé = New-Object byte[] 16
[Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($Clé)
$TexteChiffré = [System.Security.Cryptography.ProtectedData]::Protect([Text.Encoding]::UTF8.GetBytes($Texte), $Clé, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)
```

2. **Déchiffrer le texte :**

```
$TexteDéchiffré = [Text.Encoding]::UTF8.GetString([System.Security.Cryptography.ProtectedData]::Unprotect($TexteChiffré, $Clé, [System.Security.Cryptography.DataProtectionScope]::CurrentUser))
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

4. Cas Pratique : Automatisation d'une Tâche Complète

Contexte

Créer un script PowerShell pour :

- Planifier une tâche de sauvegarde.
- Chiffrer les fichiers sauvegardés.
- Générer un rapport HTML des sauvegardes.

Réseaux et sécurité

Script Exemple : Sauvegarde et Rapport

```powershell
# Script : AutomatisationSauvegarde.ps1

# 1. Configuration des variables
$Source = "C:\Dossiers"
$Destination = "D:\Sauvegardes"
$Rapport = "D:\Rapports\RapportSauvegarde.html"
$Clé = New-Object byte[] 16
[Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($Clé)
# 2. Sauvegarde des fichiers
Write-Output "Démarrage de la sauvegarde..."
Copy-Item -Path $Source\* -Destination $Destination -Recurse
# 3. Chiffrement des fichiers
Write-Output "Chiffrement des fichiers sauvegardés..."
foreach ($Fichier in Get-ChildItem -Path $Destination) {
    $Contenu = [Text.Encoding]::UTF8.GetBytes((Get-Content -Path $Fichier.FullName -Raw))
    $ContenuChiffré = [System.Security.Cryptography.ProtectedData]::Protect($Contenu, $Clé, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)
    [IO.File]::WriteAllBytes($Fichier.FullName, $ContenuChiffré)
}
# 4. Génération du rapport HTML
Write-Output "Génération du rapport..."
$RapportData = Get-ChildItem -Path $Destination | Select-Object Name, LastWriteTime, Length
$RapportData | ConvertTo-Html -Title "Rapport de Sauvegarde" | Out-File -FilePath $Rapport
Write-Output "Rapport généré : $Rapport"
# 5. Planification de la tâche
Write-Output "Planification de la tâche..."
$Trigger = New-ScheduledTaskTrigger -Daily -At "02:00AM"
$Action = New-ScheduledTaskAction -Execute "PowerShell.exe" -Argument "-File C:\Scripts\AutomatisationSauvegarde.ps1"
Register-ScheduledTask -TaskName "SauvegardeQuotidienne" -Trigger $Trigger -Action $Action -Description "Sauvegarde quotidienne automatique"
```

Réseaux et sécurité

Remoting, Sécurisation et Automatisation

Résumé

Fonctionnalité	Cmdlet / Méthode
PowerShell Remoting	`Enable-PSRemoting`, `Invoke-Command`, `New-PSSession`
Sécurisation des scripts	`Set-ExecutionPolicy`, `Set-AuthenticodeSignature`
Gestion des certificats	`Get-ChildItem Cert:\`, `Export-Certificate`
Chiffrement et déchiffrement	`ProtectedData` avec .NET
Planification de tâches	`New-ScheduledTask`, `Register-ScheduledTask`