
Exploiter, dépanner et superviser une solution d'infrastructure réseau

BTS SIO - Bloc 2 - SISR - Administration des systèmes et des réseaux



PowerShell

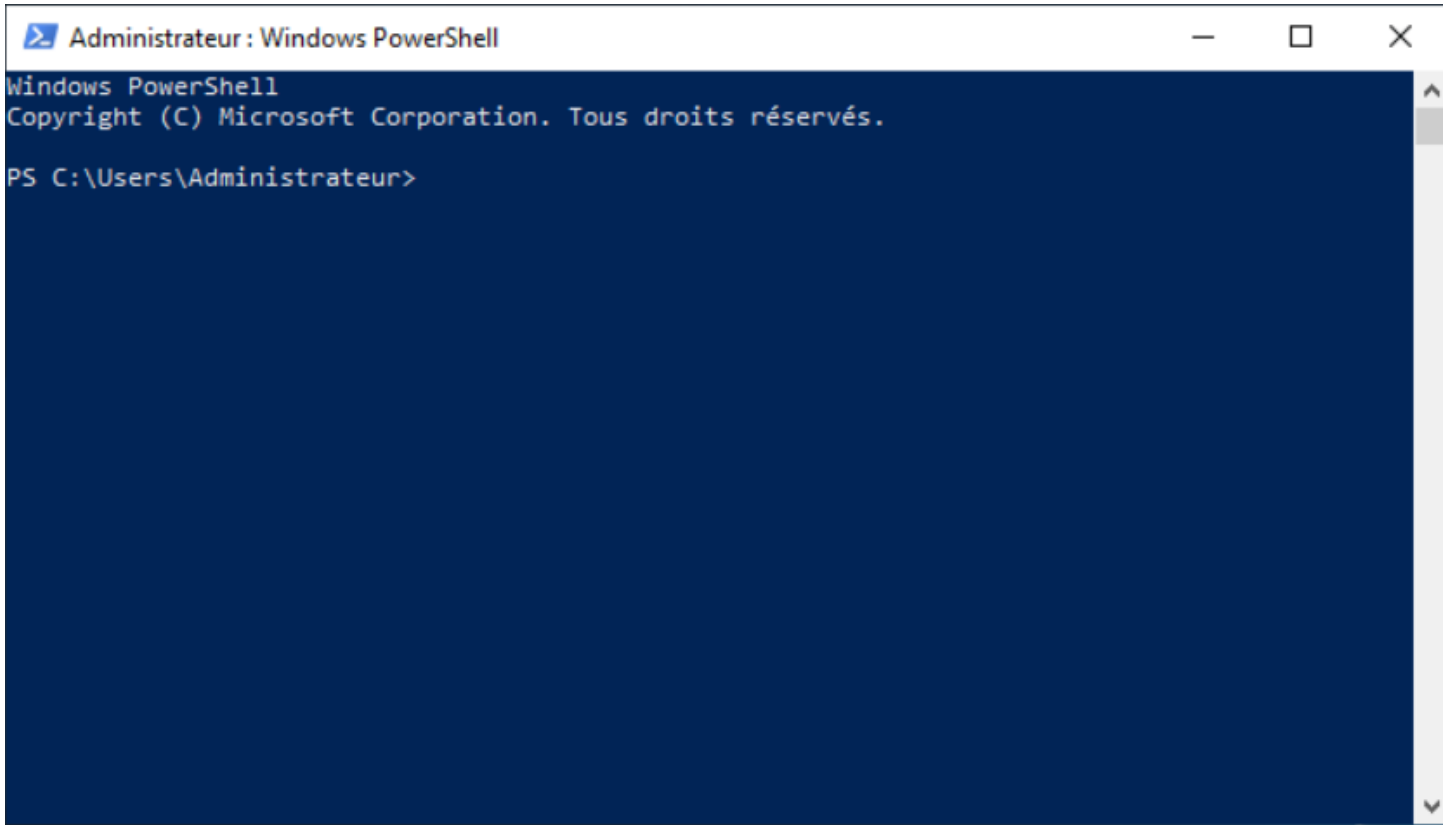
1. Introduction à Powershell	3
1.1. La console PowerShell	3
1.2. La console Windows PowerShell ISE	3
2. Aide avec Powershell	5
3. La syntaxe PowerShell	7
3.1. Les commentaires	7
3.2. Les guillemets	9
3.3. Les variables	10
4. Les boucles avec PowerShell	11
4.1. La boucle While	11
4.2. La boucle Do-While	11
4.3. La boucle For	11

1. Introduction à Powershell

PowerShell est une plateforme permettant l'exécution de scripts, il a également la fonction d'interpréteur de commandes.

1.1. La console PowerShell

La saisie des différentes commandes s'effectue par le biais de la console PowerShell. Cette dernière a un aspect graphique très proche d'une invite de commandes DOS.

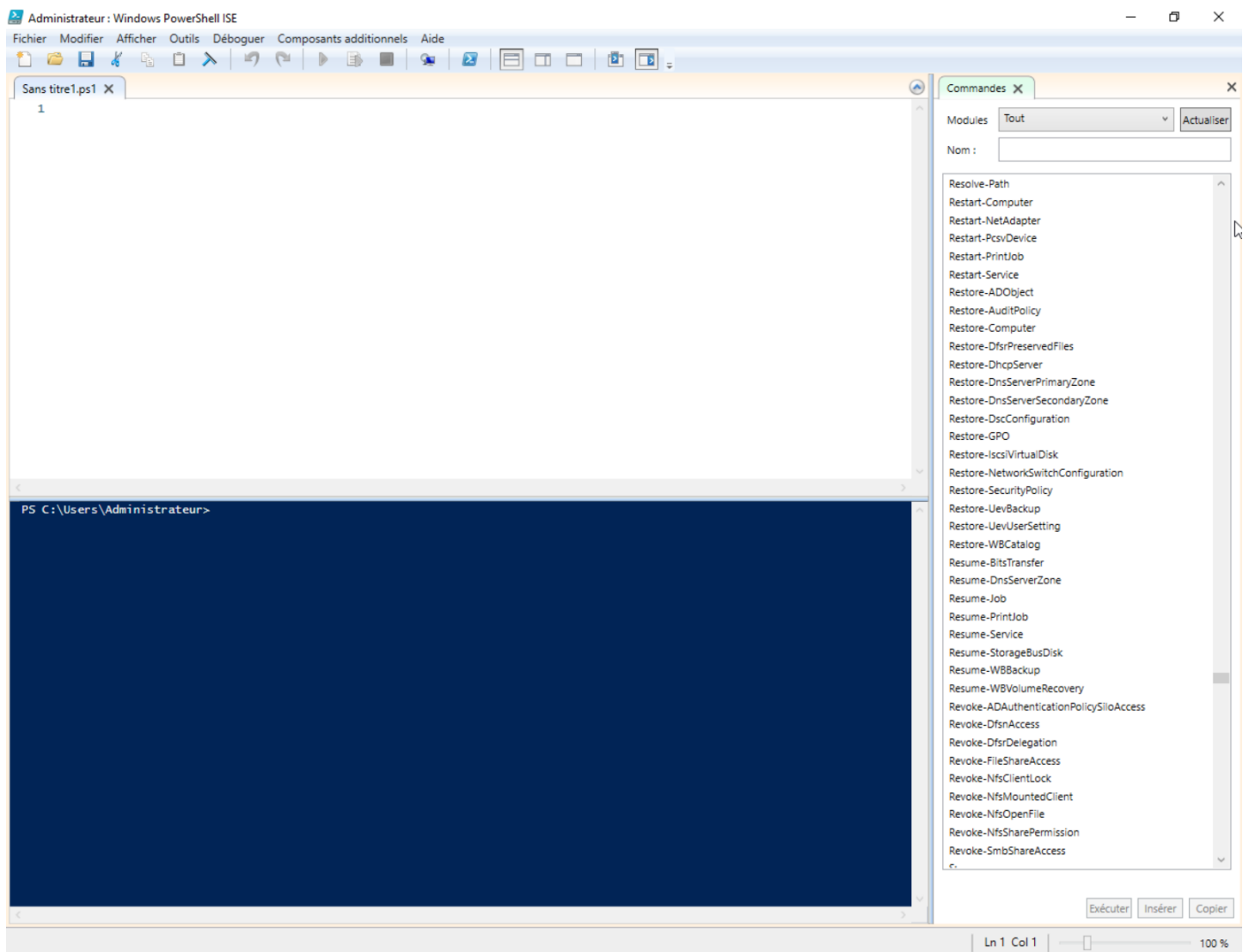


L'autocomplétion d'un chemin, d'un nom de commande, etc. peut s'effectuer à l'aide de la touche [Tab]. Pour rappel, l'autocomplétion consiste à saisir le début d'une commande ou d'un chemin, le système tente de compléter la suite.

Enfin, la combinaison [Ctrl] C ou [Ctrl][Pause] permet de mettre fin à l'exécution de l'instruction courante pour la première combinaison et de mettre fin à l'exécution de la console pour la seconde.

1.2. La console Windows PowerShell ISE

Cette console facilite la création de scripts PowerShell. Il est donc possible pour ceux qui le souhaitent et d'effectuer la création de scripts à l'aide de cet outil.



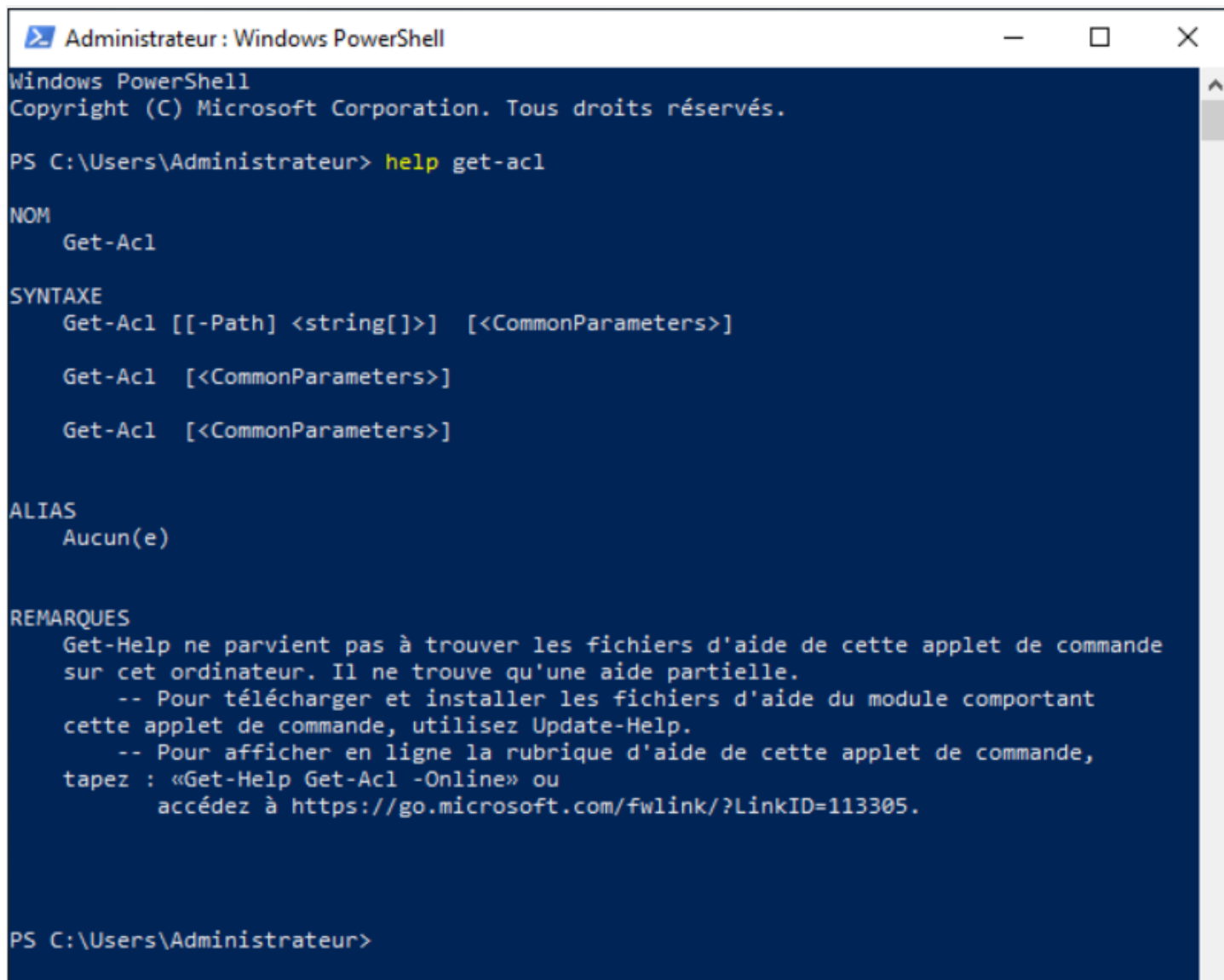
De plus, il offre plusieurs fonctionnalités intéressantes (coloration syntaxique, affichage des numéros de ligne...) mais également un débogueur.

Il est possible d'ouvrir plusieurs scripts en même temps (chacun est ouvert dans un onglet différent). Enfin, nous pouvons noter également la possibilité de tester les scripts et l'accès aux différents modules (Active Directory...). En effet, cette dernière fonctionnalité permet d'avoir accès aux différents paramètres obligatoires ou facultatifs des commandes.

2. Aide avec Powershell

Nous avons tous à un moment donné saisi le fameux `/?` dans une console DOS, ceci nous permettant de récupérer la syntaxe de la commande et un descriptif des différents commutateurs.

Avec PowerShell, la syntaxe est quelque peu différente. En effet, l'aide s'affiche en saisissant l'instruction `help` commande.

A screenshot of a Windows PowerShell window titled "Administrateur : Windows PowerShell". The window has a dark blue background with white text. The prompt is "PS C:\Users\Administrateur>". The user has entered the command "help get-acl". The output shows the command name "Get-Acl", its syntax with various parameters, and a note about help files not being found on the system. The prompt returns to "PS C:\Users\Administrateur>".

```
Administrateur : Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

PS C:\Users\Administrateur> help get-acl

NOM
    Get-Acl

SYNTAX
    Get-Acl [[-Path] <string[]>] [<CommonParameters>]

    Get-Acl  [<CommonParameters>]

    Get-Acl  [<CommonParameters>]

ALIAS
    Aucun(e)

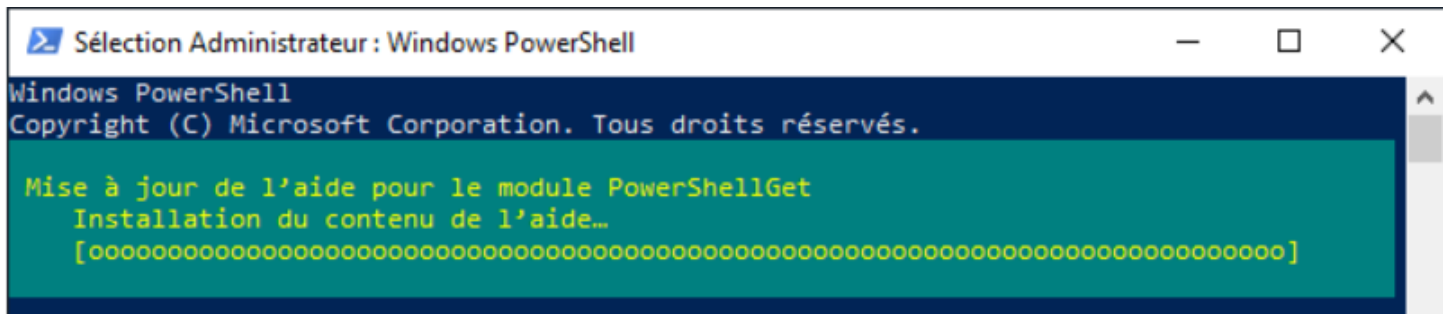
REMARQUES
    Get-Help ne parvient pas à trouver les fichiers d'aide de cette applet de commande
    sur cet ordinateur. Il ne trouve qu'une aide partielle.
    -- Pour télécharger et installer les fichiers d'aide du module comportant
    cette applet de commande, utilisez Update-Help.
    -- Pour afficher en ligne la rubrique d'aide de cette applet de commande,
    tapez : «Get-Help Get-Acl -Online» ou
    accédez à https://go.microsoft.com/fwlink/?LinkID=113305.

PS C:\Users\Administrateur>
```

Vous pouvez aussi afficher l'aide via la commande `Get-Help -Name Get-Acl`.

Les fichiers d'aide peuvent aisément être mis à jour. Il est néanmoins nécessaire d'avoir des droits d'administrateur et de lancer la console en tant qu'administrateur.

Cette opération s'effectue avec la commande `update-help`.



Cette mise à jour concerne uniquement les fichiers de la langue utilisée et du module actif. Si vous souhaitez mettre à jour le module dism alors que le module actif est AppLocker, il faut rajouter à la commande précédente le commutateur -Module dism.

Il est possible de déposer les fichiers d'aide sur un partage réseau afin que les postes qui ne sont pas raccordés à Internet puissent profiter de la mise à jour. L'opération va cette fois être scindée en deux parties (copie des fichiers sur un partage réseau et mise à jour des postes depuis ce partage réseau).

Pour effectuer cette opération, l'instruction suivante doit être utilisée : Save-Help -DestinationPath CheminUNC -Force

Enfin, il est nécessaire d'exécuter sur les postes la commande suivante : `Update-Help -SourcePath "CheminUNC" -force`

3. La syntaxe PowerShell

Une commande est constituée d'un verbe et d'un nom séparés par un tiret (-). Le verbe a pour but de décrire l'action effectuée :

- Get (obtenir une information)
- Set (effectuer une action)
- Add...

Le verbe est complété par un nom (Acl, etc.). Ce dernier constitue la commande et indique le type d'objet sur lequel l'action demandée va être exécutée.

Une liste à jour des verbes approuvés dans Windows PowerShell est accessible à cette adresse : <https://docs.microsoft.com/fr-fr/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands?view=powershell-7.1>

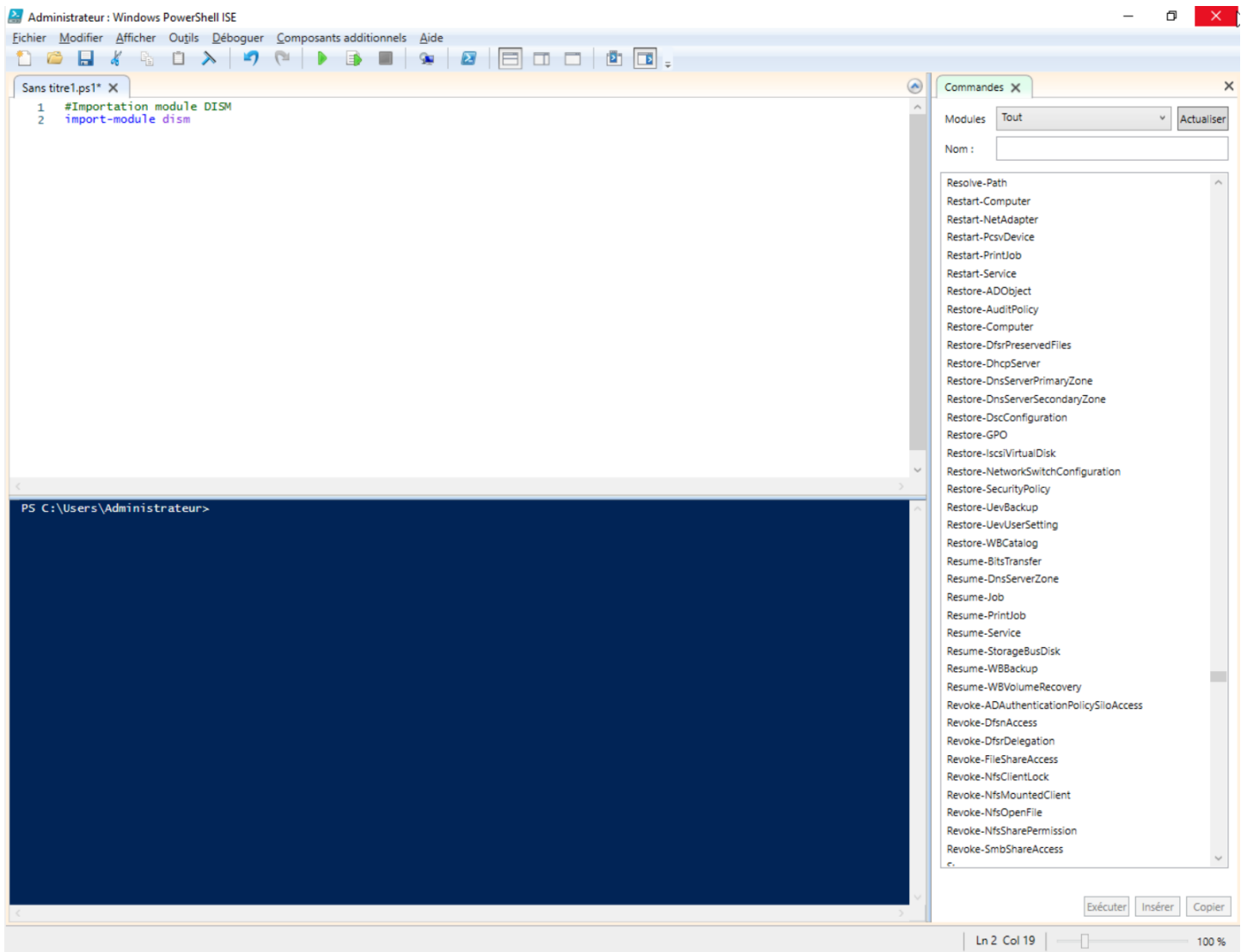
Comme pour les commandes DOS, l'analyse de la syntaxe PowerShell n'est pas sensible à la casse.

3.1. Les commentaires

Comme dans tous langages de programmation, il est possible de placer des commentaires dans les scripts.

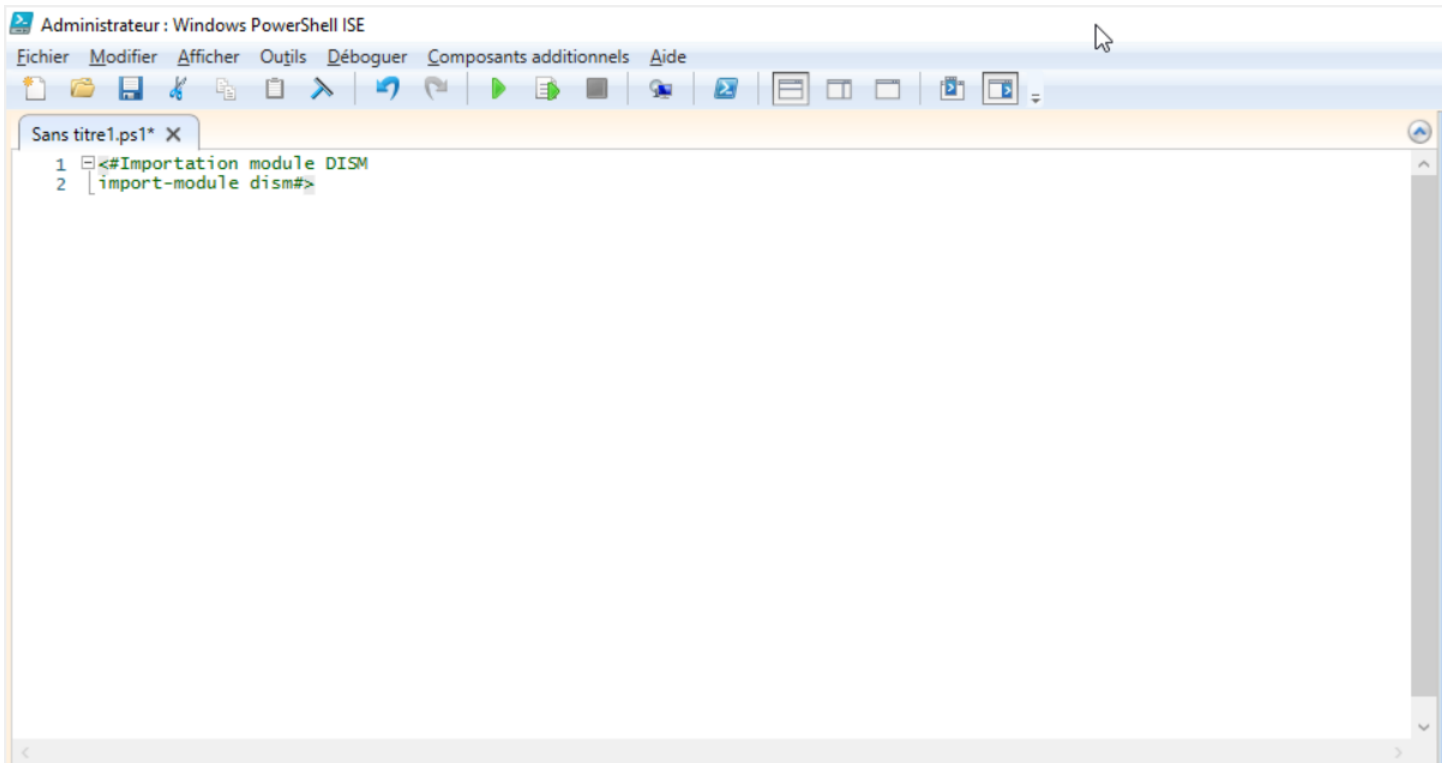
Pour différencier une instruction d'un commentaire, il faut utiliser le caractère dièse "#" en début de ligne. Ainsi la ligne est considérée comme commentaire.

De plus, l'ajout de commentaire après l'instruction peut être effectué.



Pour commenter un bloc entier, une syntaxe particulière doit être utilisée.

Les caractères `<#` doivent être insérés au début du bloc de commentaire puis, afin d'en marquer la fin, on insère les caractères `#>`.



La ligne mise en commentaire ne sera pas prise en compte par l'interpréteur de commandes.

3.2. Les guillemets

Il est nécessaire de délimiter les chaînes de caractères, ceci afin d'aider la compréhension de la syntaxe par l'analyseur syntaxique. Les guillemets (") sont généralement utilisés pour effectuer cette opération. Cette règle s'applique également à PowerShell.

Néanmoins, deux méthodes peuvent être utilisées pour délimiter une chaîne de caractères.

Les guillemets doubles " " qui permettent la substitution de variable.

```
PS C:\Users\Administrateur> $a="Vous êtes en cours en BTS SIO"
PS C:\Users\Administrateur> $a="$a, et nous découvrons Powershell"
PS C:\Users\Administrateur> $a
Vous êtes en cours en BTS SIO, et nous découvrons Powershell
PS C:\Users\Administrateur>
```

Les guillemets simples ' laissent le texte tel quel.

Reprenons la variable \$a utilisée ci-dessus et ajoutons-lui le texte : pour automatiser nos tâches..

```
PS C:\Users\Administrateur> $a='$a pour automatiser nos tâches.'
PS C:\Users\Administrateur> $a
$a pour automatiser nos tâches.
PS C:\Users\Administrateur>
```

La chaîne de caractères \$a n'a pas été prise comme variable mais comme chaîne de caractères.

3.3. Les variables

PowerShell utilise des variables comme tous les autres langages informatiques. Néanmoins, la déclaration avant l'utilisation n'est pas nécessaire. Le système va déterminer automatiquement le type de variable nécessaire (chaîne de caractère, entier...).

Par la suite, ces variables peuvent être utilisées à n'importe quel endroit du script. Afin de stocker une valeur, il est nécessaire de faire suivre la variable par un égal.

Exemple : `$a= "Bonjour !"`

Dans cet exemple, le type de la variable n'a pas été défini.

Il est possible de transformer le type de la variable (transformer une variable de type entier en chaîne de caractères).

Cette opération s'effectue de la manière suivante :

`$a=2021`

La variable est de type entier (Int32).

```
PS C:\Users\Administrateur> $a=2021
PS C:\Users\Administrateur> $a.GetType()

IsPublic IsSerial Name                                     BaseType
-----
True     True     Int32                                     System.ValueType
```

En définissant le type de variable, la même valeur est cette fois considérée comme chaîne de caractères.

```
PS C:\Users\Administrateur> [string]$a=2021
PS C:\Users\Administrateur> $a.GetType()

IsPublic IsSerial Name                                     BaseType
-----
True     True     String                                    System.Object
```

En saisissant dans la console `$a`, nous appelons cette fois la variable et le contenu de cette dernière sera affiché.

4. Les boucles avec PowerShell

Les boucles sont très utiles dans un langage de programmation pour exécuter x fois une même action.

4.1. La boucle While

Ce type de boucle permet de répéter les instructions présentes dans la boucle tant que la condition est vérifiée (vraie).

Cette boucle s'écrit de la manière suivante :

```
While condition
{
    Instruction1
    Instruction 2
    ...
}
```

4.2. La boucle Do-While

Ce type de boucle est identique à la précédente, néanmoins le test est effectué à la fin.

La syntaxe est la suivante :

```
Do
{
    Instruction1
    Instruction 2
    ...
}
While condition
```

La boucle est obligatoirement exécutée au moins une fois, le test s'effectuant à la fin.

4.3. La boucle For

Ce type de boucle offre l'avantage de pouvoir déterminer le nombre de fois où le bloc d'instructions doit être exécuté. L'exécution est effectuée tant que la valeur indiquée au départ n'est pas atteinte. Le pas d'incrément est également déclaré.

La syntaxe de ce type de boucle est la suivante :

```
For (<Valeur Initiale> ; <Condition> ; <Increment>)
{
    Instruction 1
    Instruction 2
}
```