Système d’exploitation - powershell

Get-help (help): donne l’aide sur une commande donnée :

Get-help get-help => donne l’aide au sujet de la commande get-help

Plusieurs paramètres sont possible :

-online : get-help ‘nom de la commande’ -online : donne l’aide internet

-detailed : get-help ‘nom commande’ -detailed : donne une aide plus détaillée

-examples : get-help ‘nom commande’ -examples : donne des exemples

Get-verb : donne la liste de tous les ‘’verbes’’ de commande powershell

Get-command (gcm): donne la liste de toutes les ‘’commandes’’ de powershell

gcm \*clear\* => donne la liste de commande possédant le mot ‘clear’

Set-location (sl) : défini le chemin de travail

Get-item (gi) : donne les info de l’éléments au chemin actuel

Get-ChildItem (gci) : donne les info des éléments présents dans le dossier du chemin actuel

-path (pas obligatoire à écrire si placé en premier) : spécifie le chemin du dossier

gci -path ‘’c:\...’’ : donne la liste des éléments présents dans ce dossier

gci -path ‘’c:\’’ -recurse -depth 2: donne la liste de tous les éléments présents dans le disque c à un éloignement de 2 de la racine (c :)

gci -path ‘’c:\’’ -hidden -directory -system -erroraction silentlycontinue => donne la liste de tous les dossiers (-directory) cachés (-hidden) système (-system) avec en cas d’erreur ne l’affiche pas et continue (-erroraction silentlycontinue)

gci ‘’c:\’’ -recurse -filter \*.log : va lister tous les fichiers log du disque c

Start-process powershell | runas => démarre powershell en mode administrateur

New-item (ni) : permet de créer un nouvel ‘item’ dans le chemin actuel

-itemtype : défini le type d’élément créé (par défaut : un fichier (file))

(file – directory – symboliclink – junction – hardlink)

-path : chemin à partir duquel on créera l’élément

-value : lui donner une ‘valeur’ à la création (un texte par exemple)

-force : force la création de l’élément (attention, si l’élément est déjà existant, il sera écrasé par le nouveau (perte de données, d’info…)

-confirm : demande la confirmation au lancement de la commande

ni -itemtype directory -path ‘’c:\...\toto’’ => crée le dossier toto à ce chemin-là

ni -itemtype hardlink -path ‘’c:\users\azerty\desktop\toto.txt’’ -name “blabla.txt” -target c:\users\documents\blabla.txt. => crée un hardlink toto.txt

remove-item (ri) : permet de supprimer l’élément voulu (attention ce n’est pas une mise à la poubelle, mais véritablement une suppression totale !!)

set-itemproperty (sp) : défini la ‘propriété’ de l’élément ciblé

sp ‘’c:\...’’ -name attributes -value Hidden => permet ici de ‘cacher’ l’élément ciblé (c:\...) => il existe d’autre valeur : archive/normal…

clear-itemproperty : réinitialise les ‘propriétés’ de l’élément ciblé

clear-recyclebin : vider la poubelle

test-path : test le chemin défini => et retour True / False

test-path c:\...

invoke-item (ii) : ouvre l’élément du chemin ciblé

ii ‘’c:\...’’ => va ouvrir l’élément ciblé

get-content (gc): permet de lire un fichier (d’obtenir le contenu d’un fichier dans la powershell)

gc ‘’c:\...\toto.txt’’ => va afficher dans la powershell le texte contenu dans toto.txt

gc ‘’c:\...\bidule.log’’ => permet donc d’afficher le contenu du fichier log dans la powershell

-totalcount 10 => va afficher les 10 première ligne du fichier

-tail 15 => affiche les 15 dernière ligne du fichier

-encoding utf8 => encode le contenu du fichier en UTF8 lors de sa lecture

Add-content : ajoute du contenu au fichier

Add-content ‘’c:\...\toto.txt’’ -value ‘haha hoho’ => va ajouter haha hoho au fichier texte

Set-content : ajoute du contenu au fichier en écrasant le précédant !

Select-string (sls) : recherche un mot (une chaine de caractère) dans un fichier et affiche la ligne où se trouve cette chaine de caractère

Sls ‘’c:\...\toto.txt’’ -pattern ‘’haha’’ => va chercher et afficher la ligne où se trouve « haha »

Get-history (h) : affiche votre historique des commandes effectuées (avec un numéro d’ID)

Invoke-history (ihy) : relance la commande du numéro ID voulu

Invoke-history -id 33 => relance la commande de l’historique n°33

Clear-host (clear) : efface l’écran de la powershell

Get-host : affiche les info (version…) de la powershell

Out-host : permet d’afficher à l’écran des info mais une page à la fois

Gci ‘’c:\’’ -recurse | out-host -paging => donne la liste de tous les éléments présents dans le disque c mais une page à la fois

Write-host : écrit dans le powershell

Write-host ‘’hihi hoho’’ : affichera hihi hoho dans la powershell

Out-null : effectue la commande mais n’affichera rien sur la powershell

Read-host : permet d’entrer des valeurs dans la powershell (comme un input en python)

Read-host -prompt ‘’c’est quoi ton nom ?’’ => affichera la question ‘’c’est quoi ton nom’’ et attendra que l’utilisateur rentre une valeur

Read-host -prompt ‘’c’est quoi ton nom ?’’ -asSecureString => va encrypter ce que l’utilisateur rentrera

Enable-computerrestore c:\ => active la restauration du disque c

Disable-computerrestore c:\ => désactive la restauration du disque c

Checkpoint-computer : crée un point de restauration de l’ordinateur

Checkpoint-computer -description ‘’blabla’’ -restorePointType application\_install => crée un pt de restauration

Get-computerrestorepoint : donne les point de restauration existant (listé avec un num de séquence)

Restore-computer -restorepoint 1 => restaure l’ordi avec le point de restauration ayant comme num de séquence 1

Get-random : donne un nombre aléatoire

-minimum -5 => un nombre aléatoire dans un range commençant par –5

-maximum 29 => finissant par 29

-setseed 12 => défini une ‘seed’ de 12 => nombre aléatoire toujours identique selon la ‘seed’ choisie

Get-random 10,20,30 => retourne aléatoirement un nombre sur les 3 donnés (10 ou 20 ou 30)

Get-psdrive : donne tous les disques présents sur l’ordi

Gci env:\ => donne tous les éléments enfants de env :

Get-item env:\users => va donner dire qui est l’utilisateur

New-psdrive DOCS -psprovider filesystem -root c:\Users\...\documents\ -description ‘’disque temporaire vers le dossier documents’’ => crée le drive DOCS qui est relié au dossier Documents => ne sera visible qu’en powershell !!

Remove-psdrive DOCS => élimine le drive DOCS

Get-process => donne la liste des process en cours d’exécution

Get-service => donne la liste des services de l’ordi

Get-service -displayname SPOOLER => récupère les info du service SPOOLER

Stop-service -name SPOOLER

Start-service -name SPOOLER

Set-service -name SPOOLER -startuptype Automatic => défini le mode de démarrage du service

Get-alias => donne la liste de tous les alias

New-alias ed -value notepad.exe => crée un alias nommé ‘ed’ qui une fois appelé démarre notepad

Ed toto.txt => lance notepad avec le fichier toto.txt (qu’il aura préalablement créé si nécessaire !)

Set-alias -name ed -value powershell => modifie l’alias ‘ed’ afin qu’il démarre la powershell

Export-alias => exporte tous les alias

Export-alias ed -as script => exporte l’alias ‘ed’

Pipeline => | => permet de chaîner plusieurs action l’une après l’autre…

Gci | sort-object length => récupère les éléments ‘enfants’ et les affiche trié selon leur longueur

Gci | get-member => donne toutes les propriétés existantes de gci

Get-service | sort-object status => trie les services selon leur statuts

Gci -recurse -path c:\ -system -erroraction silentlycontinue | measure-object

=> compte le nombre de fichier système sur tout le disque c

Gcm | sort-object commandtype -descending | formtable -autosize convertto-html | ou-file -filepath c:\...\desktop\command.html -encoding utf8

=> crée un fichier html sur le bureau qui contient toutes les commandes existantes en powershell

Gci -recurse -directory -system -path c:\ -erroraction silentlycontinue | where length -gt 200KB | sort length -desc

=> affiche tous les dossiers systeme de c : dont leur longeur (poid) est plus grand que 200kb et les trie selon leurs poid (du plus gd ou plus petit)

Get-process | where cpu -gt 5 | out-host -paging

=> donne tous les process qui ont un temps cpu plus long que 5 sec

Gci -path c:\ -recurse -filter \*.txt -erroraction silentlycontinue | compress-archive -destinationpath c:\...\desktop\archive.zip -compressionlevel optimal

=> récupère tous les fichiers txt du disque c et les compresse dans une archive .zip que l’on place sur le bureau

Get-service | where status -eq running | select displayname | foreach-object {write-host $\_.displayname -foregroundcolor (get-random darkblue, yellow, grey…)}

=> affiche uniquement les noms des services qui sont en cours d’utilisation et les affiche chacun d’une couleur aléatoire parmis celles définies

Les scripts POWERSHELL fichier en ‘.ps1’

# permet un commentaire sur une ligne

<#

commentaire

sur

plusieurs

ligne

#>

$nomdevariable => création d’une variable (signe $ obligatoire devant le nom de la variable)

${nom de variable avec espace} => signe {} obligatoire en cas d’utilisation de nom de variable avec espace

$a = 32 => la variable à comme valeur 32

$a.gettype() => récupère le type de la variable (ici int32)

Clear-variable a => vidange la variable sans la détruire ni en retirer le type

New-variable => permet de créer une nouvelle variable => attention une constante ne sait ni changer de valeur ni être détruite ! (au contraire d’une readonly => peut être modifiée avec -force ou détruite)

Opérateurs de comparaisons et logiques

|  |  |
| --- | --- |
| Opérateurs | Description |
| -eq, -ceq, -ieq | Égalité |
| -ne, -cne, -ine | Inégalité |
| -gt, -cgt, -igt | Plus grand que |
| -ge, -cge, -ige | Plus grand ou égal |
| -lt, -clt, -ilt | Plus petit que |
| -le, -cle, -ile | Plus petit ou égal |
| -contains, -ccontains, -icontains | Contient |
| -notcontains, -cnotcontains, -inotcontains | Ne contient pas |

Entre 2 comparaisons =>

-and

-or

-not

-xor

=> attention de bien mettre entre parenthèse () chaque comparaison

(10 -eq 15) -and (1,2,3 -contains 1)

(False) -and (True) => False

2 possibilités équivalentes :

$age -in [6,7]

[6,7] -contains $age

Structure conditionnelle :

If

Elseif => correspond au elif en python

Else

Les boucles

Foreach 🡺 boucle finie d’une collection

2 méthodes d’utilisation

En ligne de commande powershell 🡺 foreach-object{}

En utilisation dans un script 🡺 foreach(){}

Exemple comparatif :

gci c:\ | foreach-object {$\_.name} 🡺 ligne de commande dans powershell avec des |

foreach($element in gci c:\){$element.name} 🡺 dans un script powershel

exercice : récupérer tous les fichiers log de l’ordi, et afficher leur nom et chemin d’accès en vert si leur poids est < 5Mo et en rouge si > 5 Mo

$element = gci c :\ -recurse -filter \*.log -erroraction silentlycontinue -file

foreach($item in $element){

if ($item.length -gt 5Mb) {

write-host $item.fullname -foregroundcolor Red

}

Else {

Write-host $item.fullname -foregroundcolor Green

}

}

For 🡺 boucle finie

For (début de boucle ; condition de fin ; pas)

For ($freq=1000 ; $freq -le 4000 ; $freq +=300) {

[system.console]::Beep($freq,100)

}

🡺 ici il y aura un beep à la fréquence 1000Hz jusqu’à 4000Hz par pas de 300Hz

While(assertion) {action} 🡺 boucle indéfinie 🡺 comme en python, nécessite dans son action le critère de fin

Do{action}

While(assertion) 🡺 boucle indéfinie qui passe une première fois obligatoirement dans l’action avant de vérifier la condition 🡺 tant que la condition soit vraie

Do{action}

Until(assertion) 🡺 idem que ci-dessus 🡺 a moins que la condition soit vraie (opposé)

Exercice : suite de syracuse : on part d’un entier. Si celui-ci est pair on le divise par 2, s’il est impaire on le multiplie par 3 et on additionne 1. 🡺 La suite se terminera toujours par une boucle 4. 2. 1…

Le temps de vol => durée de la suite

Altitude max => nombre le plus haut atteint

Temps de vol en altitude => nombre de valeur plus haute que le nombre de départ

[int]$nbr = read-host -prompt ”Entrer un nombre”

[int]$tpsvol = 0

[int]$max = 0

[int]$tpsvolalt = 0

[int]$val = $nbr

While ($val -ne 1){

If ($val % 2 -eq 0){

$val = $val / 2

}

Else {

$val = $val\*3+1

}

$tpsvol += 1

If ($max -lt $val){$max = $val}

If ($val -gt $nbr){$tpsvalalt +=1}

Write-host ($val)

}

Write-host “le temps de vol est de :’’$tpsvol

Write-host ‘’l’altitude max est de :’’$max

Write-host ‘’le temps de vol au dessus de $nbr est de :’’$tpsvolalt

Les tableaux

!!!! 🡺 la taille d’un tableau en powershell est figé (pas dynamique comme dans python) 🡺 on ne sait pas le modifier en taille 🡺 Powershell « modifiera » le tableau en en créant un nouveau qui écrasera le précédant…

$tableau = 64, ‘Hello’, 3.5, ‘Wolrd’

$tableau = @(64, ‘Hello’, 3.5, ‘Wolrd’)

$tableau = @() => tableau vide

$un, $deux, $trois, $quatre = $tableau

🡺 $un prendra la valeur d’indice 0 du tableau (ici 64), $deux la valeur ‘Hello’…

🡺 si plus de donnée dans le tableau que de variable ($un, $deux…) alors la dernière variable prendra toute les dernières valeurs du tableau

$tableau = (1..7) 🡺 1,2,3,4,5,6,7

[int[]]$tableau = 12,64,35 🡺 on force le typage du tableau

$tableau += 62 🡺 on ajoute 62 à la fin du tableau (voir remarque début paragraphe) 🡺 12,64,35,62

$tableau[1] 🡺 retourne la valeur d’indice 1 🡺 64

$tableau[1..3] 🡺 retourne les valeur d’indice de 1 à 3

$tableau[-1] 🡺 retourne la dernière valeur du tableau

$demo = @(2,4,6,7)

$final = $demo | where-object (($\_ -ne 7))

🡺 retire la première instance de 7 du tableau demo 🡺 $final => 2,4,6

Exercice commencé au cours à finir pour le prochain cours… 🡺 crible d’Erastothene (voir la définition sur wikipedia) ☺

[int]$val = read-host -prompt ”valeur max”

$primes = @{}

“les nombres premiers entre 1 et $val sont : »

« 1 »

Foreach ($i in 2..$val){

If($primes[$i] -eq $null){

$i

$primes[$i] = $true

For($j = $i\*$i ; $j -le $val ; $j += $i) {

$primes[$j]=$false

}

}

}