TARTALOM

Random hasznos dolgok

Ctrl+Space: kilistázza a lehetséges paramétereket a terminálban és kódolás közben is
Get-Help
Get-Command
Get-Process: kilistázza a futó processeket
param mindig az elején legyen!!!
komment: #
többsoros komment: <# …#>

Operator	Example	Description
&	& script.ps1	Executes a script or command.
.	. script.ps1	Executes a script in the current scope.
()	($a + $b) * $c	Ensure that expressions are evaluated in a specific order
$( )	$(Get-Date)	Subexpression operator; runs the command in a subexpression.
`	"This is a backtick `n"	Escape character; used to include special characters in strings.
..	$a = 1..10	Create an array with a range of integer from 1 to 10.

Hasznos linkek

https://netwrix.com/
https://lazyadmin.nl/

Futtatás

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
-> el kell fogadni a felugró ablakot
.\fájlnév vagy **powershell -ep bypass -f .\fájlnév.ps1**

Kiírás

echo
-> Format-List, Format-Table, Format-...
Write-Output
Write-Host
-> formátumban több lehetőség, pld: -ForegroundColor [Color] -BackgroundColor [Color] “szöveg”
$változó
//kiírni a tömb minden elemét: $tömb vagy $tömb.ForEach({Write-Host “Szöveg” $_ }) ha kísérőszöveg is kell

Paraméterek

alapvetően az $args tömbben vannak
param (
$param1,
$param2=’default’,
[Parameter(Mandatory)]$param3,
[int]$param4
…
)
meg lehet adni default paramétereket, így ha ezek nincsenek megadva a futtatásnál, akkor a default értékkel fut le, ha meg van adva, akkor felülíródik -> második paraméter
meg lehet adni, hogy egyes paraméterek megadása nélkül ne fusson le a program, hanem ha nincs megadva, akkor kérje be külön -> harmadik paraméter
meg lehet adni konkrét típust -> negyedik paraméter

Fájl beolvasása

Soronként tömbbe: Get-Content
$tömb = Get-Content -Path .\fájl
//Az indexelés 0-tól van
További lehetőségek:

- Első x sor beolvasása:
  Get-Content -Path .\fájl **-TotalCount x**
- Utolsó x sor beolvasása:
  Get-Content -Path .\fájl **-Tail x**
- utolsó sor
  (Get-Content -Path .\fáj)**[-1]**
- x. sor:
  (Get-Content -Path .\fájl)**[x]**
  (Get-Content -Path .\fájl **-TotalCount x**)**[-1]** //gyorsabb
- Összes fájl beolvasása egy mappából
  Get-Content -Path .\mappa*

- Bizonyos típusú fájlok beolvasása egy mappából
  Get-Content -Path .\mappa* **-Filter '*.txt'**
- Egy darab stringbe való beolvasás
  Get-Content -Path .\fájl -Raw //gyors, replacenél előnyös

Változók

Command	Description
`$a = "Hello"`	Assign a value to a variable
`$a, $b = "Hello","Bye"`	Multiple variable assignment
`$range = 1..10`	Create an array with a sequence
`$_`	Get current pipeline object
`$null`	Null value
`[type]$var`	Declared typed (int, string) variable
`$global:var`	Assign variable in global scope

Változóba való beolvasás: $v = Read-Host

Stringek

Összefűzés, szétválasztás

Operator	Example	Description
-split	'one,two,three' -split ','	Splits a string into an array based on a delimiter.
-join	('one','two') -join ','	Joins array elements into a single string.

Konvertálás Int-be

[int]$string = "100"

Replace (+"delete")

Replace()

-konkrét egyezés
-NINCS regex
-case-sensitive
$string.Replace("régi","új")

-replace

-VAN regex
-case-sensitive
$string -replace “régi”, “új”

TÖRLÉSRE IS JÓ
pld. $string -replace “törlendő”, “”

Dátum

https://lazyadmin.nl/powershell/get-date/

Tömbök

linkek: contains és annak változatai,

Deklarálás:

$tömb = @(‘elem1’, ‘elem2’, ‘elem3’)
üres tömb: $tömb = @()
többdimenziós tömb:
$tömb = @(
@(‘elem11’, ‘elem12’),
@(‘elem21’, ‘elem22’)
)
tömb előre meghatározott hosszal: $tömb = [Object[]]::new(x)

Kiírás

Ha valami kísérőszöveg kell minden sor elejére: $tömb.ForEach({Write-Host “Szöveg” $_ })
Vagy csak simán **$tömb**

Index

$tömb[index]

Elem hozzáadása a tömbhöz (~append)

$tömb += “elem”

Elem eltávolítása a tömbből

$tömb = $tömb | Where-Object { $_ -ne “elem”}

Tartalmaz-e egy elemet (true/falset ad vissza) contains

Nem case-senstive: $tömb -contains ‘elem’

Case-sensitive: $tömb -ccontains ‘elem’
Nem tartalmazza: $tömb -notcontains ‘elem’

Megszámolja hányszor van benne egy elem a tömbben

$hányszor = $tömb -eq ‘elem’

Elemek száma

$tömb.Length

Típus meghatározása

[int[]]$tömb = 1,2,3 //int-ekből álló tömb

Subset

$subset = $tömb[3..6]

Összefűzés

-join
Pld:
$tömb = @(‘egy’,’két’,’há’)
$tömb -join ‘-’
Kimenet: “egy-két-há”

Két lista összefűzése

$összefűzött = $lista1 + $lista2

Reverse/megfordítás

[Array]::Reverse($tömb)

Filter/kiválogatás

$tömb | Where-Object {feltétel (amikre igaz, azokat válogatja ki)}

Listává alakítás

Lista deklarálása: $lista = New-Object System.Collections.ArrayList
[System.Collections.ArrayList]$lista = $tömb

Különböző elemek kiválogatása két listából

Compare-Object

Csoportosítás

GroupObject

Csoportosítja a tömbben lévő elemeket, és meg is számolja, hogy miből hány darab van

```
# Define a list of product codes with different cases
$productCodes = @("AB123", "ab123", "CD456", "cd456", "AB123", "CD456", "ef789", "EF789")

# Group the product codes case-sensitively
$groupedProductCodes = $productCodes | Group-Object -Property { $_ } -CaseSensitive

# Display the results
$groupedProductCodes | ForEach-Object {
    Write-Output "Group: $($_.Name)"
    $_.Group | ForEach-Object {
        Write-Output " - $_"
    }
}
```

Kiválogatás

Select-Object

van egy csomó szuper paramétere, lehet válogatni egyszer előforduló elemeket, bizonyos indexeket, elsőt, utolsót stb

Sorbarendezés

Sort-Object

“Struktúra”

Igazából egy tömb objektumokkal
$struktúra = @(
[obejktumnév]@{Név=”xy”;Foglalkozás=”szakács”}
…
)
ugyanúgy le lehet kérdezni mondjuk az első objektumot: $struktúra[0]
első objektumban lévő ember neve: $struktúra[0].Név
ugyanilyen hivatkozással meg is lehet változtatni az adatokat

Szótár/Dictionary/Hashtable

`$hash = @{`
`key1 = 'value1';`
`key2 = 'value2';`
`}`
	Create hash table

`$hash.key1`	Access hash key
`$hash.key2 = 'new value'`	Assign value to hash key
`$hash.Add('key3', 'value')`	Add key-value pair
`$hash.Remove('key2')`	Remove key-value pair

`$obj = [PSCustomObject]@{`
`Prop1 = 'Val1';`
`Prop2 = 'Val2'`
`}`
	Custom object

$myHashtable = @{

"Fruit1" = "Apple"

```
    "Fruit2" = "Orange"

    "Fruit3" = "Cherry"

}
```

Classok

```
class Osztaly
{
    #Random tulajdonság
    [típus]$Név

    #Előre definiált tulajdonság, felül lehet írni (~paraméterek)
    [típus]$Név = 'Valami'

   #Függvény visszatérési értékkel
   [típus] Függvény()
   {
       #kód, pl készítsen el egy szöveget/urlt az egyik tulajdonsággal
   }

    #Visszatérési érték nélkül void típusú, pl írja ki a szöveget/nyissa meg az urlt
}
```

Új class beli változó létrehozása: **$változó = [Osztály]::new()**
Hivatkozás tulajdonságra: **$változó.Név**

Operátorok

Változókhoz (csak a szoki)

Operator	Example	Description
=	$a = 5	Assigns the value on the right to the left.
+=	$a += 3	Adds the right operand to the left and assigns it.
-=	$a -= 2	Subtracts the right operand from the left and assigns it.
*=	$a *= 4	Multiplies the left operand by the right and assigns it.
/=	$a /= 2	Divides the left operand by the right and assigns it.

–	-$var	Negates the value of the operand.
++	++$var	Increments the operand's value by 1.
—	–$var	Decrements the operand's value by 1.

Összehasonlítás

FONTOS: egyenlőségnél a változó/tömb van jobb oldalt, mert különben megszámolja, hogy a szám hányszor van benne a változóban

Operator	Example	Description
-eq	5 -eq $a	Returns `True` if both operands are equal.
-ne	5 -ne $a	Returns `True` if operands are not equal.
-gt	$a -gt 5	Returns `True` if the left operand is greater.
-lt	$a -lt 5	Returns `True` if the left operand is less.
-ge	5 -ge $a	Returns `True` if the left operand is greater or equal.
-le	5 -le $a	Returns `True` if the left operand is less or equal.

Operator	Example	Description
-match	$string -match "foo"	Returns `True` if the string matches the regex pattern.
-notmatch	$string -notmatch "foo"	Returns `True` if the string does not match the regex pattern.
-replace	$string -replace "foo", "bar"	Replaces text that matches a regex pattern.
-like	$string -like "foo*"	Returns `True` if the string matches the wildcard pattern.
-notlike	$string -notlike "foo*"	Returns `True` if the string does not match the wildcard pattern.

Típusok

Operator	Example	Description
-is	$object -is [Type]	Returns `True` if the object is of the specified type.
-isnot	$object -isnot [Type]	Returns `True` if the object is not of the specified type.

Logikai

Operator	Example	Description
-and	(5 -gt 3) -and (6 -gt 2)	Returns `True` if both conditions are true.
-or	(5 -gt 7) -or (8 -gt 2)	Returns `True` if any of the conditions are true.
-not	-not (5 -eq 5)	Reverses the logic of its operand.
!	!(3 -eq 4)	Short form of `-not`.

Bemenet-kimenet szabályozása

Operator	Example	Description
>	Get-Process > processes.txt	Redirects standard output to a file.
>>	Get-Process >> log.txt	Appends standard output to a file.
2>	Get-Command xyz 2> errors.txt	Redirects error output to a file.
2>>	Get-Command xyz 2>> errors.log	Appends error output to a file.
2>&1	Get-Process 2>&1 alloutput.txt	Redirects error stream to the succes stream
*>	Get-Process *> alloutput.txt	Redirects all streams to a file

Vagy Out-File (több lehetőség, pl -NoNewline, -Encoding, -Confirm stb)

✨Flow control statements✨

LEHET BREAKELNI juhuuu

If-else

if (utasítás) {...} else {...}

Switch

switch ($v) {}
($v) {}

For

for ($i=0; $i -lt n; $i++){

…
}

Foreach

$tömb.ForEach({utasítás})
pld. kiírni a tömb minden elemét: $tömb.ForEach({Write-Host “Szöveg” $_ })
aktuális paraméter: $_
foreach ($elem in $tömb) {
…
}
utasítás minden elemre: **ForEach-Object {...}**

While, do-while, do-until

while (statement) {...}
do {...} while (statement)
do {...} until (statement)

Függvények

PRO-TIP: érdemes ige-főnév típusú elnevezéseket használni

```
function függvénynév {
        param (
                [típus]$paraméternév1,
                [típus]$paraméternév2
                …
        )
        #kód
}
```

Függvényhívás: függvénynév -paraméternév1 paraméter1 -paraméternév2 paraméter2 …

Csővezeték (pipeline) :D

Ugyanaz, mint bash-ben

Előző órai munka

```
//1.feladat
#1 feladat -help
param(
   [switch]$help
)
function Show-Help{
```

```
    "This"
}
if($help) {Show-Help; exit}
//2.feladat
#2.feladat Minden sor ami [ERROR]-al kezdődik azt írassuk ki
param(
    [switch]$Analyse
)
function AnalizeLogs($file){
    $regex = "(^\[ERROR\])"
    foreach($line in Get-Content .\$file){
        if($line -match $regex){
        $line
        }
    }
}
if($Analyse) {AnalizeLogs $args[0]; exit}
//3.feladat
#3. azok a txt fileok amik régebbiek 30 napnál, kerüljenek törlésre
$curr_date=Get-Date
$curr_date.AddDays(-30)
Get-ChildItem . "*.txt" -Recurse |
    Where-Object {
        if (($curr_date -ge  $($_.CreationTime))){
            Remove-Item -Path $_ -Force
        }
```