PowerShell

If space in between the words use “”.

Set-alias list get-childitem (will only in memory so need to set aliases each time we open the cmd. Other option is to export before exit the console window and import from the file again)

**ISE (Integrated Scripting Environment)**: Is like editor has intelligence use crl space.

**Get-Command** is to get all the command

Get-Help Get-Command also a short cut **Get-Command -?**

Set-Location C:\dir1path

**Pipe**: use for further working on the result. E.g get all the child items where length is greater than 100kb

| should be the last char of the command

Set-Location C:\Windows

Get-ChildItem **|** where-Object { **$\_**.Length –gt 1kb } **|** Sort-Object Length

Get-ChildItem **|** where-Object { $\_.Length –gt 1mb } | Sort-Object Length **|** Format-Table -Property Name, Length, IsReadOnly

**S\_** is current object/like lambda.

Set-Location env:

Get-ChildItem (show children of environment variables)

**Variables**: variables start with **$. $myVar = "my variable abc"**

And we can use now as $myVar or $myVar.Length, $myVar.GetType() etc.

For string we can use combination of ‘ ’ “ @ for multiple lines “ some ’text’ abc”

String interpolation: We can also use varibales inside the strings using $. “value of variable is $myVar”

We can also use expression in side strings like “the sum of 10 and 15 is **$(**10+5**)**”

Also can do formatting like C#: “The total {0}” –f $myVar

Wild cards (\*? [] for range): “Mystring” –like “Mys\*” => Ture

For comments use **#**

[System.Int32] $myInt = 40

Variables types are Strings, Arrays, Hash Tables

Everything in PowerShell is Dot net object.

**Comparison for integers**:

-eq Equals

-ne not equal to

-lt Less Than

-gt Greater then

-le Less than or equal to

-ge Greater than or equal to

**Comparison for Strings**:

-Like Like wildcard pattern matching

-NotLike Not Like

-Match Matches based on reqular expressions

-NotMatch Non-Matches based on reqular expressions

Calculations are like any other language $var = 3\*11

**Arrays**: $MyArray = "a1", "a2"

$MyArray[0]

$numbers = 1,4,6, 99

$numbers -notcontains 10 => False

**Hash Tables**: to hold list of values like arrays

$hashTbl = **@{ "Key" = "value"**

"MS" = "Microsoft"; "AU" = "Australia" **}**

$hashTbl["AU"]; $hashTbl.**Remove**("AU"); $hashTbl.**Contains**("AU")

**Built in variables**: $false, $true, $NULL, $pwd (current directory)

$home (home directory), cd $home

**Branching if/else**: we can also do nested branching or **switch** block

$var1 = 2;

if ($var1 -eq 1) { cls; "if branch" }

else { cls; "else branch"}

$varx = 10;

**switch** ( $varx ) {

1 { 'One' ; break } ; 10 {"Ten"};

default {"Hundred"} }

Looping:

for ($i = 0; $i -le 10; $i++ ) { $i; };

$i = 1;

**while** ($i -le 5) { "`$i = $i"; $i = $i + 1; }

**Do While**: $i = 6; do { "`$i = $i" ; $i = $i + 1; }

while ($i -le 5);

Example:

$myArrayX = 10,20,30,40, 50;

for ($i = 0; $i -lt $myArrayX.Count; $i++ ) {

"value at $i = " + **$myArrayX[$i]** }

**Script Blocks**: any code with in { code }, we can store script block in variable & to run use & before the variable.

$myScriptBlockVar = { 'Allah oh Allah ..'}

for ($i=0; $i -lt 4; $i++ ) { **&$myScriptBlockVar;** }

**Parameters with script blocks using args**: script blocks are just like functions in C#.

**$qa = {** **$question = $args[0];**

**$answer = $args[1];**

Write-Host "Question: $question" "Answer: $answer"  **}**

**&$qa** "whats your name?" "Khan"

$qaParam = { **param ($question , $answer = "The question has no answer" )**

Write-Host "Question: $question" "Answer: $answer" }

&$qaParam "whats your name?" "Jadoon!"

&$qaParam –**answer** "Jadoon!!" - **question** "what’s your name?"

&$ qaParam **-a** "Jadoon!!" **-q** "what's your name????-----"

Add two numbers

$AddNo = { param (**[int] $x, [int] $y**);

return $x + $y; }

**&$AddNo 30 8**

**Variables Scope**: if you change the value inside the script block it’s only for that block. As it make a local copy inside the script block/function. But you can get parent value inside the block using –scope 1. For this you can also use **$global:myVar**.

$myVar = 10;

& { $myVar = 25; Write-Host "var value inside function: $myVar"

Write-Host "Parent var: " (**Get-Variable myVar -ValueOnly -Scope 1** )

}

Write-Host "var value: $myVar"

**Build a CI/CD pipeline from PowerShell**:

Using Team Module to Add new builds, releases, remove build, add git repository