2020

PowerShell Fast Track (Cheat Sheet)

Sukhija, Vikas

http://TechWizard.cloud

POWERSHELL SCRIPTING

FOR

NON SCRIPTERS AND SCRIPTERS

This small book is not a regular one; it is full of small scripts that can be used by System administrators during their day to day IT operations. This book is not for freshers but for experienced IT administrators that want to use scripting & do IT automations. Just copy paste the code blocks from the book/links to make simple to complex scripts.

Note: - Please check for spaces while copy pasting the codes.

Author: Vikas **Website:** http://TechWizard.cloud

Table of Contents

1. PowerShell Basics

Let us start with basic stuff & touch base quickly on Variables, Loops, if else, Switch and functions. These are heart of any scripting language and will assist you in creating simple to complex scripts.

I will not talk about PowerShell versions, What is PowerShell?, On what all platforms it can be used ?, Get, SET commands. If you are looking for in-depth details on those topics than this book is not right for you.

This book is for IT admins that want to quickly generate simple to complex scripts. I have tested this approach with many of my students and it worked great for them.

Not everyone is from programming background and good at creating code but following the approach described in this book you will be able to quickly create your own scripts and automate IT systems/processes.

1.1. Variables & Printing

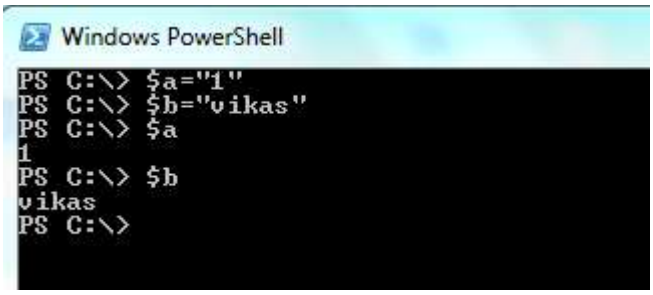
To begin with we need to understand the basics which include variables/arrays. Every variable in PowerShell starts with a dollar $ sign.

For example: -

$a = “1”

$b = “Vikas”

When you will type $a & $b values will be displayed.



Now you can use **write-host** to print this to screen

Input: PS C:\> Write-host $a

Output: 1

Input: PS C:\> Write-host $b

Output: vikas

TIP: make thumb rule of Quotes when assigning values to variables as shown above in two examples.

Change the foreground color of what is being displayed by Write-host.

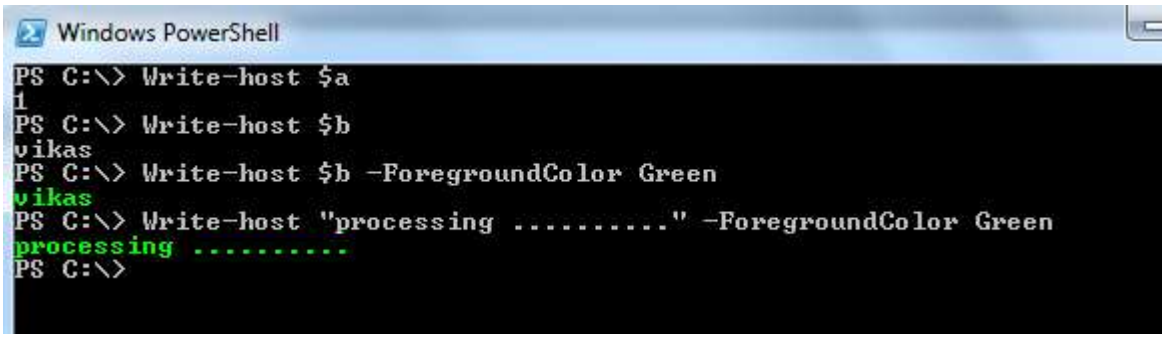
Input: PS C:\> Write-host $b -ForegroundColor Green

Output: vikas

Input: PS C:\> Write-host "processing ....." -ForegroundColor Green

Output: processing .....

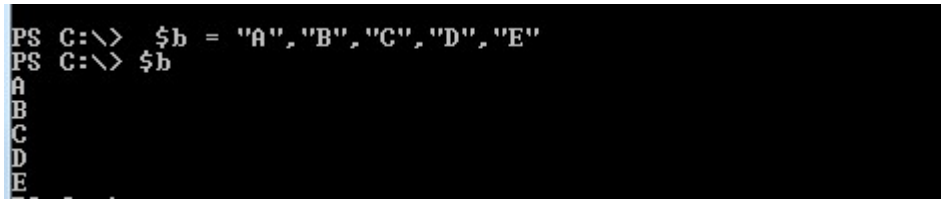
PS C:\>



I am not illustrating arrays separately in this book.

In PowerShell, arrays can also be defined in the same way as variables. Below are the examples:

$b = "A","B","C","D","E"



$c= @("server1","server2")

Dynamic Array can be defined as @() , these type we will use in examples inside this book to add elements in the array and generate reports.

$d = @()

1.2. If Else/ switch

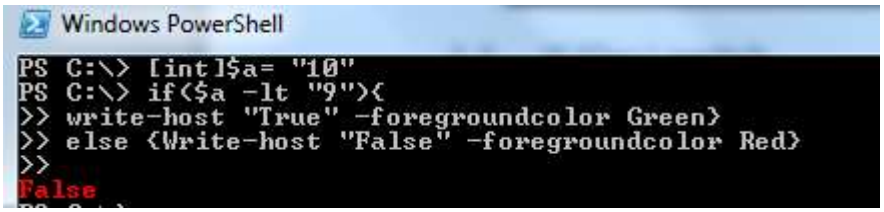
IF ELSE is a condition based processing, it's the bases of any scripting language. If some condition is true you need to process something otherwise process some other thing.

Here I have illustrated two examples; first we have defined variable value as 10, then using conditional operators mentioned in 1.2.1 and if else statements we checked if it's greater than 9 or if it's less than 9.

-gt means greater than and -lt means less than, do not worry we will quickly go thru them in next sub section.

```
[int]$a= "10"

if($a -gt "9")
{
write-host "True" -foregroundcolor Green
}else {
Write-host "False" -foregroundcolor Red
}
```


```
[int]$a= "10"

if($a -lt "9"){
write-host "True" -foregroundcolor Green
}else {
Write-host "False" -foregroundcolor Red
}
```



1.2.1 Conditional / Logical Operators

Below is the list of conditional /Logical operators which you will use in your everyday scripts. Without these, many of the scripting operations will not be possible.
These will always be used in comparison, if else statements as shown in above parent section.

-eq	Equal
-ne	Not equal
-ge	Greater than or equal
-gt	Greater than
-lt	Less than
-le	Less than or equal
-like	Wildcard comparison
-notlike	Wildcard comparison
-match	Regular expression comparison
-notmatch	Regular expression comparison
-replace	Replace operator
-contains	Containment operator
-notcontains	Containment operator

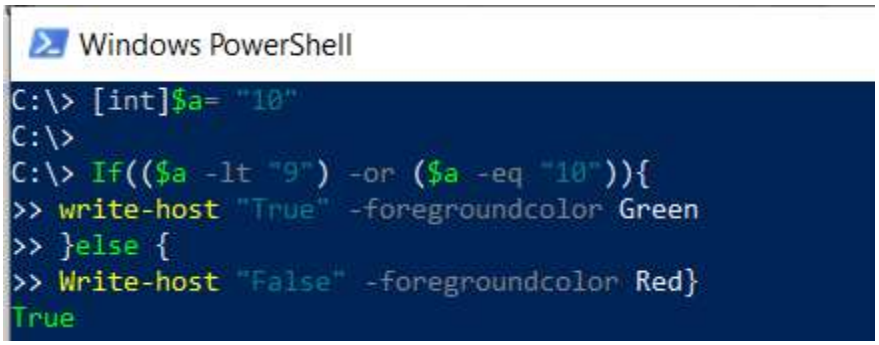
Logical operators

-and	Logical And
-or	Logical Or
-not	Logical not
!	Logical not

Logical operators will be used when you want to combine the conditions, lets update on above example.

We will print true if value of variable $a is less than 9 or equals to 10.
Here we have combined two conditions, as its OR operator, so TRUE will be retuned if one of them matches. Here second condition matches as value is equal to 10.

```
[int]$a= "10"

If(($a -lt "9") -or ($a -eq "10")){
write-host "True" -foregroundcolor Green
}else {
Write-host "False" -foregroundcolor Red}
```



If you will use AND operator, then both conditions should match if you want to return TRUE, which will not happen in above case.

Not is for negation, I generally use it very less, my experience say it causes human error if you are in hurry and do not think thru 😊

1.3. Loops

There are two main Loops in any Scripting language & that's true with PowerShell as well:

For Loop & While Loop

1.3.1 For –Loop

There are three different iterations of For loops:

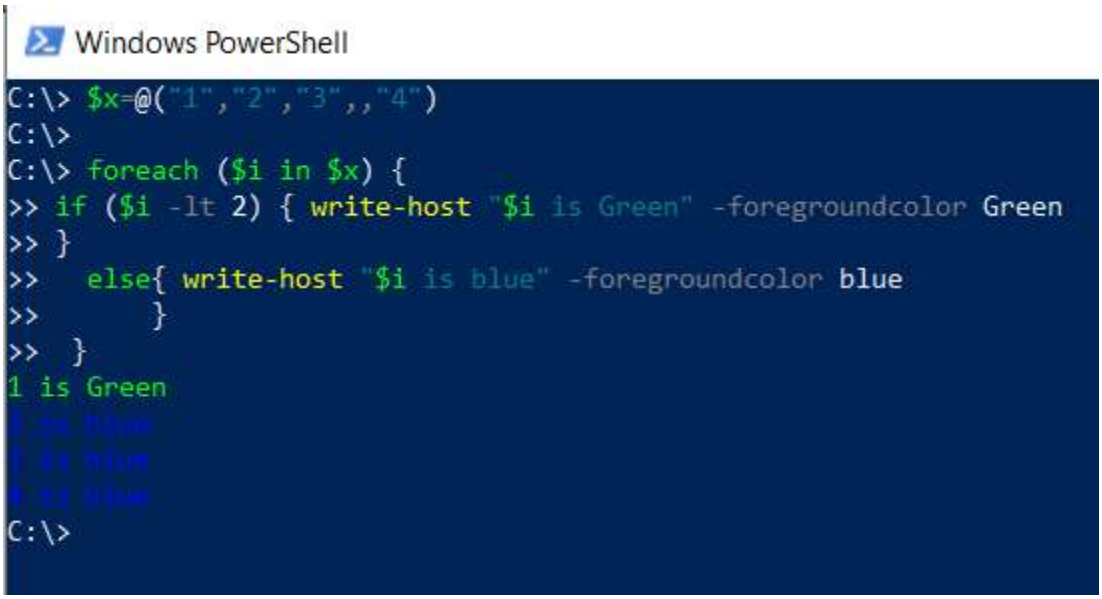
ForEach
ForEach-Object
For

Now let's look at examples to differentiate between the three.

**ForEach: You need to specify foeach $variable in $collection as shown in below example -** foreach ($i in $x)

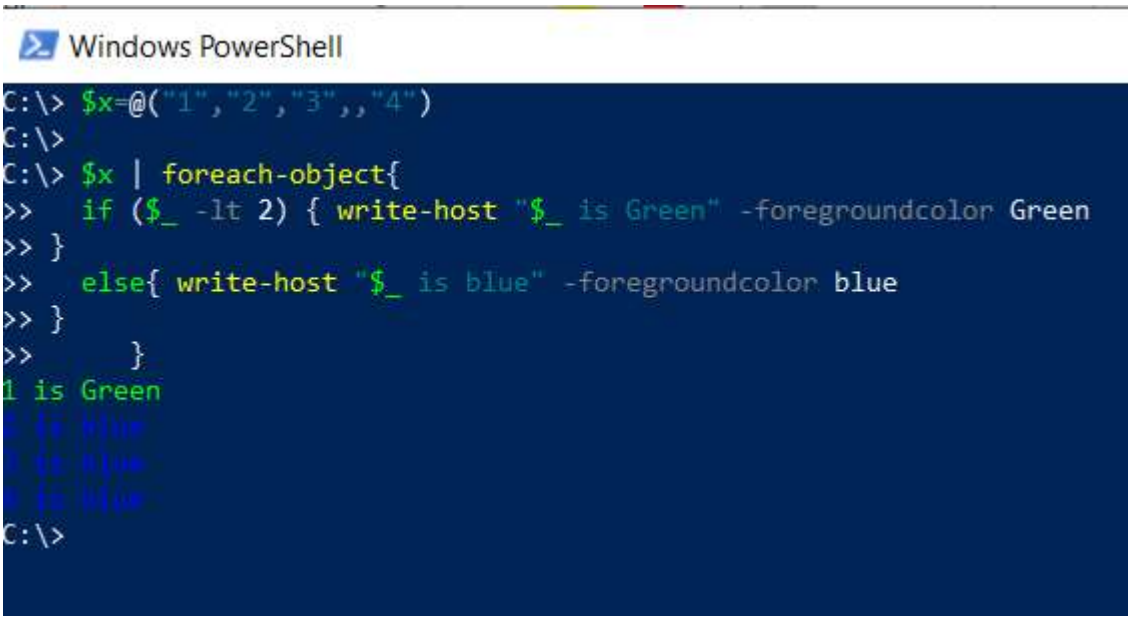
```
$x=@("1","2","3",,"4")
```

```
foreach ($i in $x) {
        if ($i -lt 2) { write-host "$i is Green" -foregroundcolor Green
        }
        else{ write-host "$i is blue" -foregroundcolor blue
    }
}
```



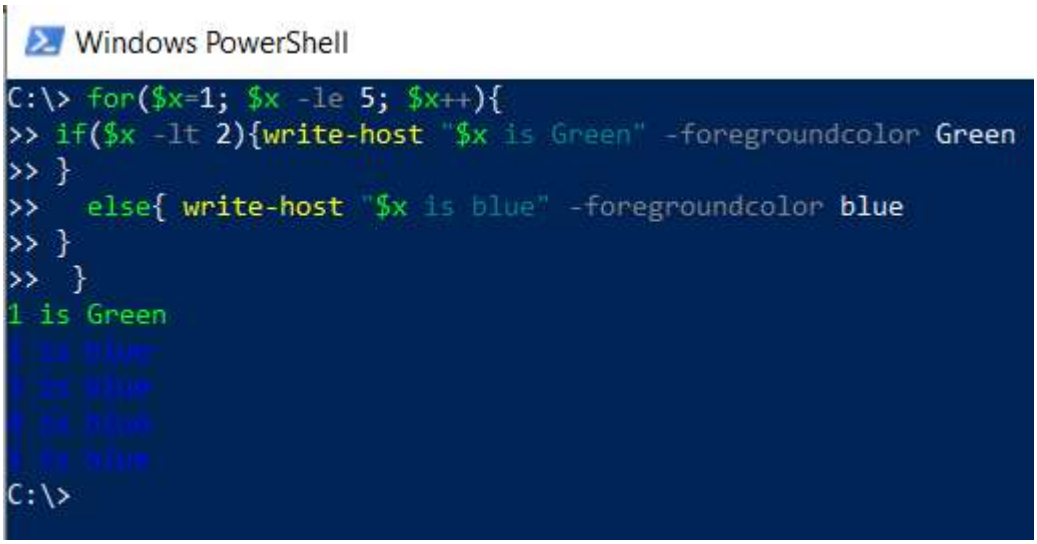
ForEach-Object: You will use PIPE with collection to achieve the same thing
As shown below - $x | foreach-object

```
$x=@("1","2","3",,"4")

$x | foreach-object{
        if ($_ -lt 2) { write-host "$_ is Green" -foregroundcolor Green
        }
        else{ write-host "$_ is blue" -foregroundcolor blue
        }
    }
```



For: This is the one which you will remember from you school days, I have not used it much and seen very less usage across the community.

```
for($x=1; $x -le 5; $x++){
        if($x -lt 2){write-host "$x is Green" -foregroundcolor Green
        }
        else{ write-host "$x is blue" -foregroundcolor blue
        }
 }
```



1.3.2 While –Loop

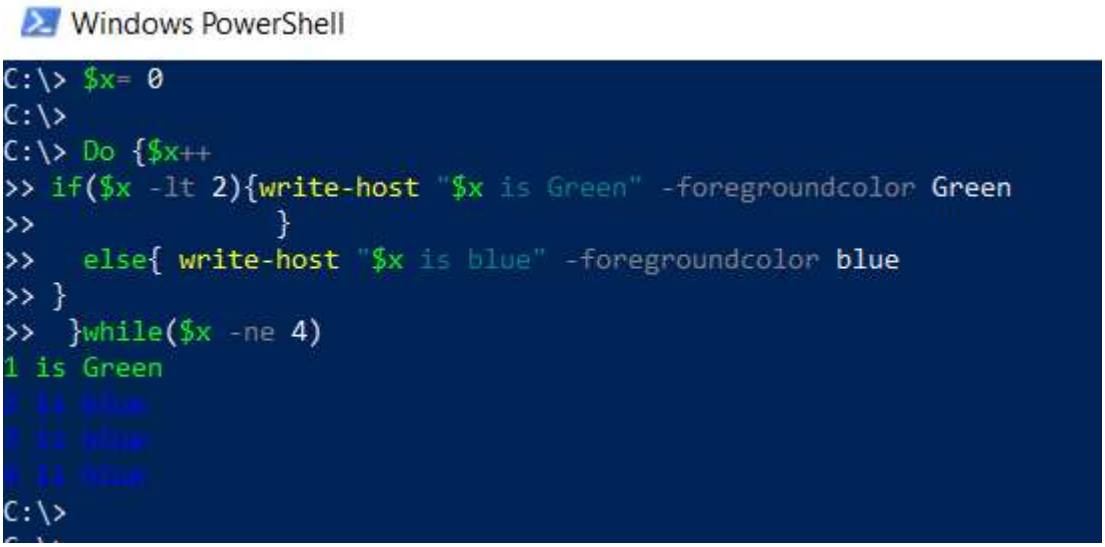
While Loop is different as it lasts till the condition is true, see below examples:
While loop also has two iterations

Do-While
While

Below are the examples:
For **Do-While** we are doing something until some condition is met.
In example below variable x = 0 and inside the variable we are incrementing its value until its not equal to 4.

Note: we are doing the thing first and match the condition later.

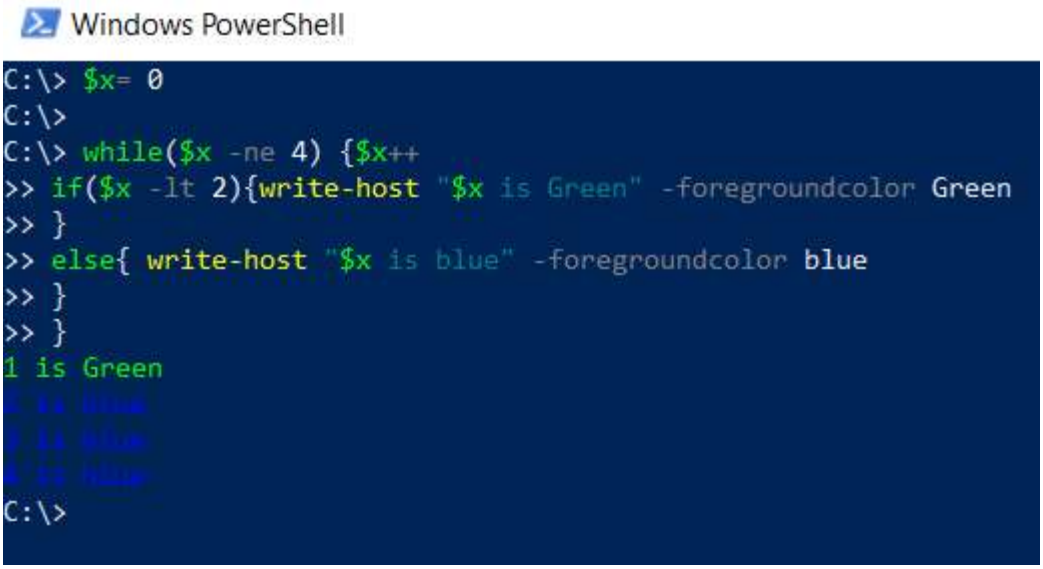
```
$x= 0

Do {$x++
if($x -lt 2){write-host "$x is Green" -foregroundcolor Green
            }
          else{ write-host "$x is blue" -foregroundcolor blue
          }
 }while($x -ne 4)
```



For **While** as well we are doing something until some condition is met.
In example below variable x = 0 and inside the variable we are incrementing its value until its not equal to 4.

Note: we are checking first and doing the thing after that.

```
$x= 0

while($x -ne 4) {$x++
if($x -lt 2){write-host "$x is Green" -foregroundcolor Green
        }
        else{ write-host "$x is blue" -foregroundcolor blue
        }
}
```



1.4. Functions

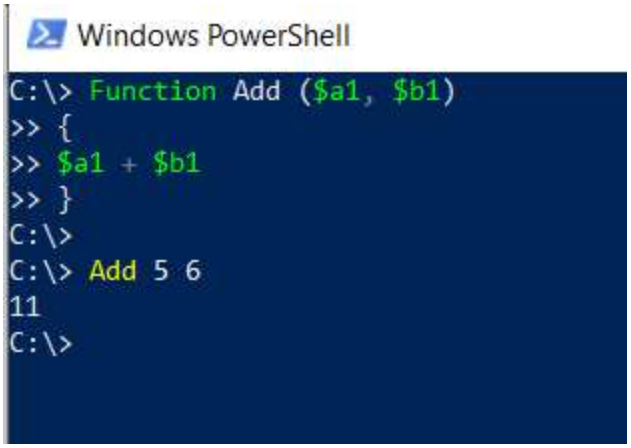
Functions are those entities, ones defined you can call them again & again in any part of the script & avoid repetitive code.

Create Function

```
Function Add ($a1, $b1)
{
$a1 + $b1
}
```

Call function

```
Add 5 6
```


Windows PowerShell
C:\> Function Add ($a1, $b1)
>> {
>> $a1 + $b1
>> }
C:\>
C:\> Add 5 6
11
C:\>

2. Date & logs

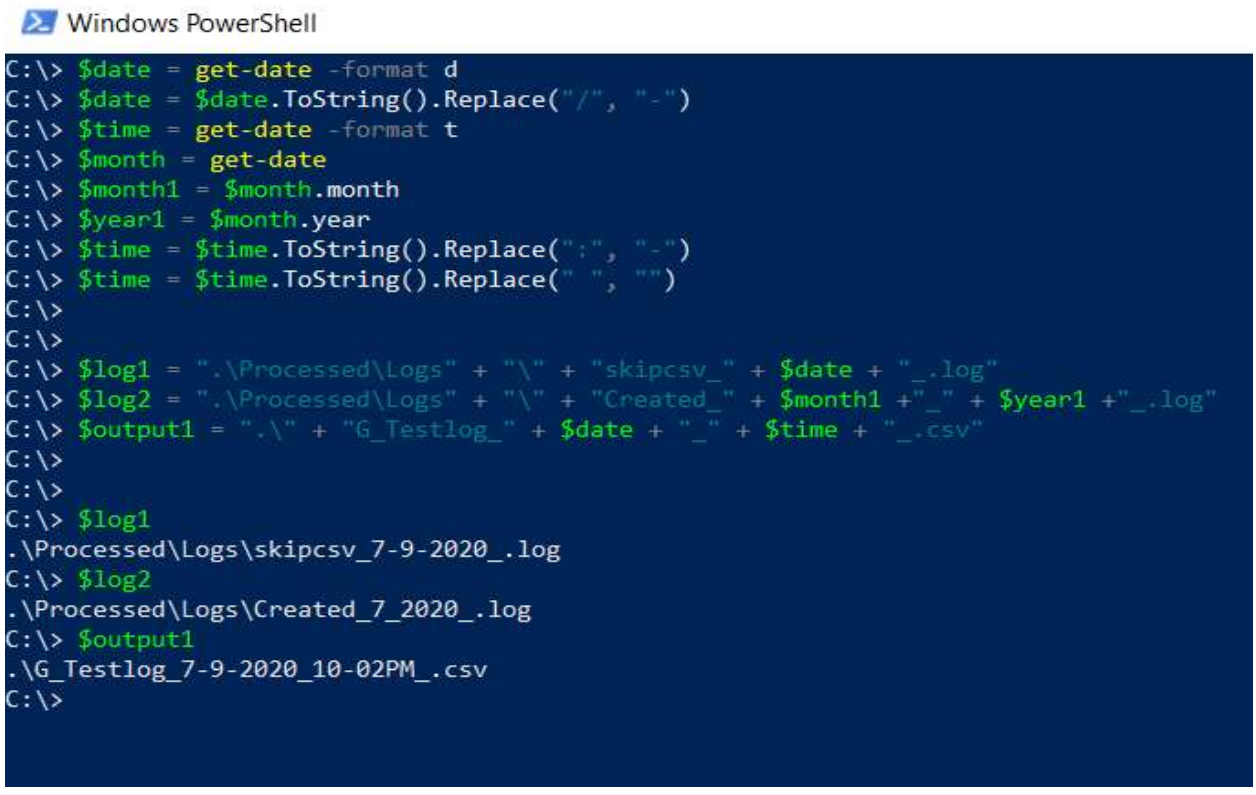
2.1. Define Logs

For creating the scripts, it is essential to Create Log files and if you create it dynamically based on date & time than it would be great, so here is the code that can be used. Below code can be used in the beginning of the script to define log file paths.

```
$date = get-date -format d
$date = $date.ToString().Replace("/", "-")

$time = get-date -format t
$month = get-date
$month1 = $month.month
$year1 = $month.year
$time = $time.ToString().Replace(":", "-")
$time = $time.ToString().Replace(" ", "")
```

Examples:-

```
$log1 = ".\Processed\Logs" + "\" + "skipcsv_" + $date + "_.log"
$log2 = ".\Processed\Logs" + "\" + "Created_" + $month1 +"_" + $year1 +"_.log"
$output1 = ".\" + "G_Testlog_" + $date + "_" + $time + "_.csv"
```



Note: - .\ always define the current working folder.

You can also use below log function, if you follow me, you will find in most of my scripts.

```
function Write-Log
{
  [CmdletBinding()]
  param
  (
    [Parameter(Mandatory = $true,ParameterSetName = 'Create')]
    [array]$Name,
    [Parameter(Mandatory = $true,ParameterSetName = 'Create')]
    [string]$Ext,
    [Parameter(Mandatory = $true,ParameterSetName = 'Create')]
    [string]$folder,

    [Parameter(ParameterSetName = 'Create',Position = 0)][switch]$Create,

    [Parameter(Mandatory = $true,ParameterSetName = 'Message')]
    [String]$Message,
    [Parameter(Mandatory = $true,ParameterSetName = 'Message')]
    [String]$path,
    [Parameter(Mandatory = $false,ParameterSetName = 'Message')]
    [ValidateSet('Information','Warning','Error')]
    [string]$Severity = 'Information',

    [Parameter(ParameterSetName = 'Message',Position = 0)][Switch]$MSG
  )
  switch ($PsCmdlet.ParameterSetName) {
    "Create"
    {
      $log = @()
      $date1 = Get-Date -Format d
      $date1 = $date1.ToString().Replace("/", "-")
      $time = Get-Date -Format t

      $time = $time.ToString().Replace(":", "-")
      $time = $time.ToString().Replace(" ", "")

      foreach ($n in $Name)
      {$log += (Get-Location).Path + "\" + $folder + "\" + $n + "_" + $date1
+ "_" + $time + "_.$Ext"}
      return $log
    }
    "Message"
    {
      $date = Get-Date
      $concatmessage = "|$date" + "|   |" + $Message +"|  |" + "$Severity|"
```

```
      switch($Severity){
        "Information"{Write-Host -Object $concatmessage -ForegroundColor
Green}
        "Warning"{Write-Host -Object $concatmessage -ForegroundColor Yellow}
        "Error"{Write-Host -Object $concatmessage -ForegroundColor Red}
      }

      Add-Content -Path $path -Value $concatmessage
    }
  }
} #Function Write-Log
```

To create log file you can simply use it as below: (do not forget to create folders logs and report first)

```
$log = Write-Log -Name "Name-Log" -folder "logs" -Ext "log"
```

For creating CSV file for report purposes you can use it as:

```
$Report1 = Write-Log -Name "MAM-Report" -folder "Report" -Ext "csv"
```

To write information to log file you can use

```
Write-log -Message "Connect to Intune" -path $log
```

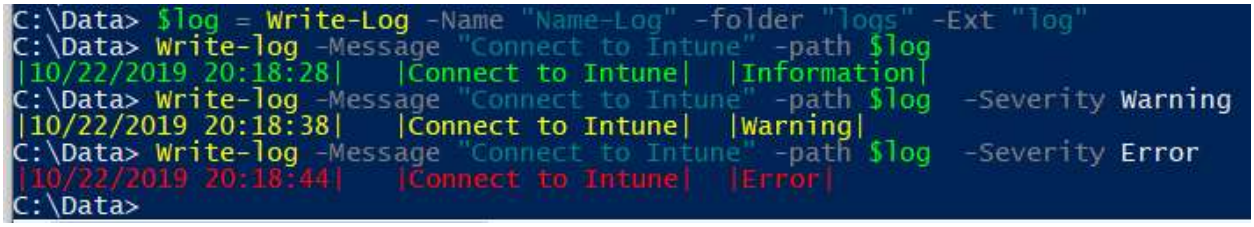
To write warning to log file you can use as

```
Write-log -Message "Connect to Intune" -path $log -Severity Warning
```

To write error to log file you can use as

```
Write-Log -Message "Error loading Modules" -path $log -Severity Error
```

Screenshot of the operation:

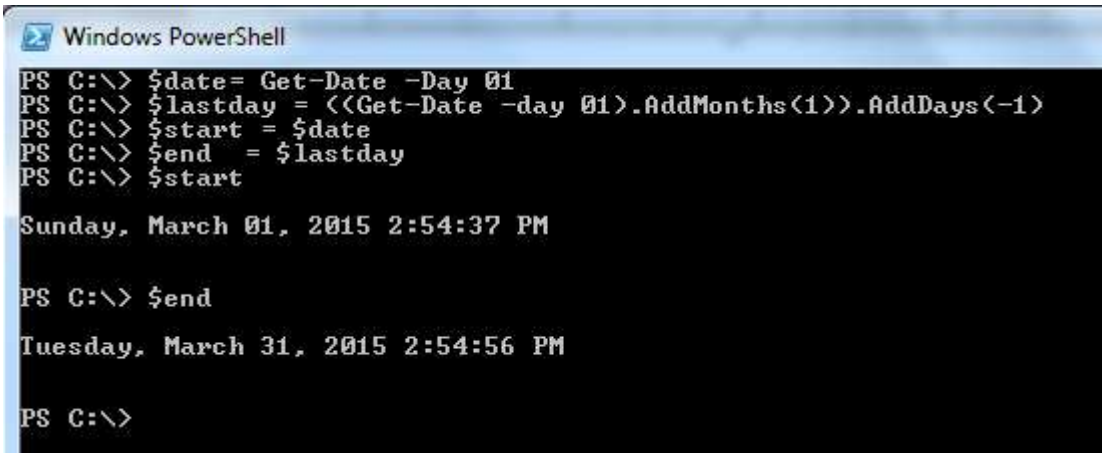


2.2. First day & Last day of Month

Get-date will get you current date and time, we have to manipulate it according to needs of our script as shown below.

To get first day & last day of the Months you can use below Code

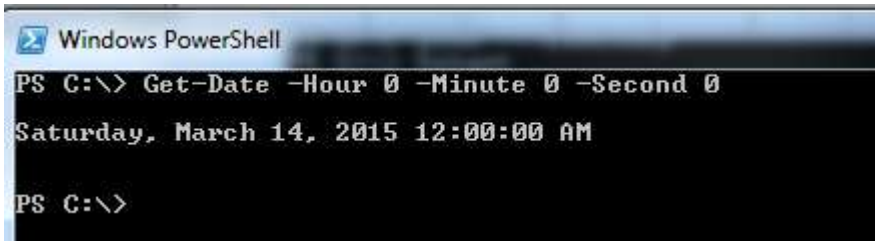
```
$date= Get-Date -Day 01
$lastday = ((Get-Date -day 01).AddMonths(1)).AddDays(-1)

$start = $date
$end  = $lastday
```



2.3. Midnight

To get the midnight

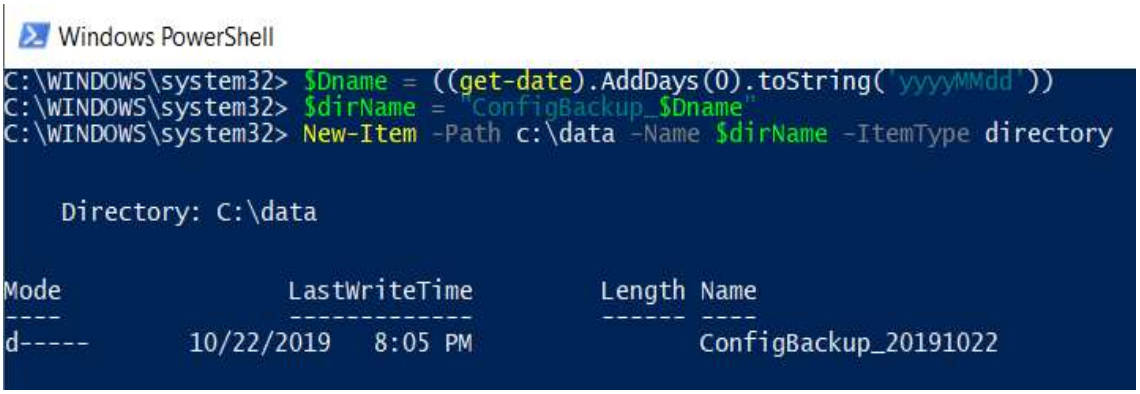
Get-Date -Hour 0 -Minute 0 -Second 0



2.4. Create Folders based on Date

This is the example to create folders based on Date, This will be helpful in situations where you want to place data inside new folder every day.

```
$Dname = ((get-date).AddDays(0).toString('yyyyMMdd'))
$dirName = "ConfigBackup_$Dname"

New-Item -Path c:\data -Name $dirName -ItemType directory
```



2.5. Recycle Logs

As logs will get on accumulating so there is a need of recycling of logs so that drives don't get fill. This is important for all your scripts for which you have enabled logging.

Here is the code that can be used:

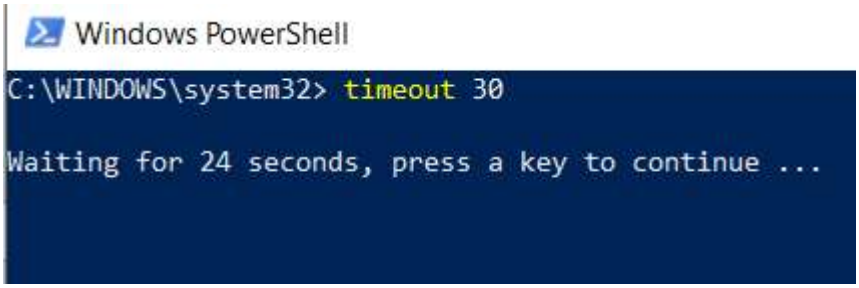
Where $limit defines the number of days, anything older than 60 will be deleted as per below code.

```
$path1 = "c:\data\logs"
$limit = (Get-Date).AddDays(-60) #for recycling
Get-ChildItem -Path $path1 |
Where-Object -FilterScript {$_.CreationTime -lt $limit} |
Remove-Item -Recurse -Force
```

2.6. Progress bar

If you want to add nice progress bar to your scripts then here is the code you can add for waiting instead of simple timeout command

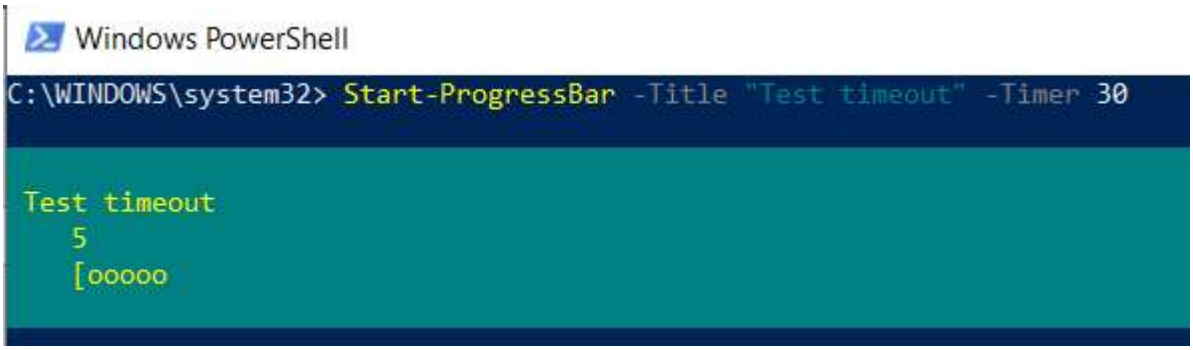
Simple timeout



Progress Bar function

```
function Start-ProgressBar
{
  [CmdletBinding()]
  param
  (
    [Parameter(Mandatory = $true)]
    $Title,
    [Parameter(Mandatory = $true)]
    [int]$Timer
  )

  For ($i = 1; $i -le $Timer; $i++)
  {
    Start-Sleep -Seconds 1;
    Write-Progress -Activity $Title -Status "$i" -PercentComplete ($i /100 *
100)
  }
}
```

Start-ProgressBar -Title "Test timeout" -Timer 30



3. Input to your scripts

Now I will share the code that can act as input for your scripts.

3.1. Import CSV

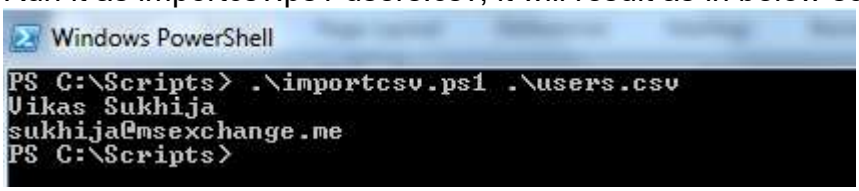
By using below code you can import csv file as an input to your scripts.

```
$data = import-csv $args[0]    #this means that you will specify csv when    executing script

foreach ($i in $data) {
Write-host  $i.user #reading column user
Write-host  $i.email #reading column email
   }
```

To test it create a csv file with two columns

	A	B
	user	email
	Vikas Sukhija	sukhija@msexchange.me

Run it as importcsv.ps1 users.csv, it will result as in below screen.

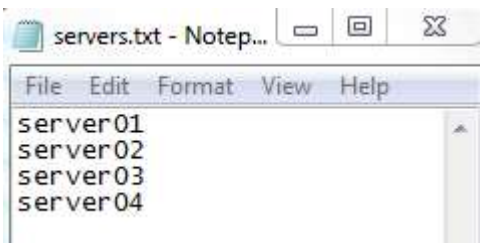


3.2. Import from text file

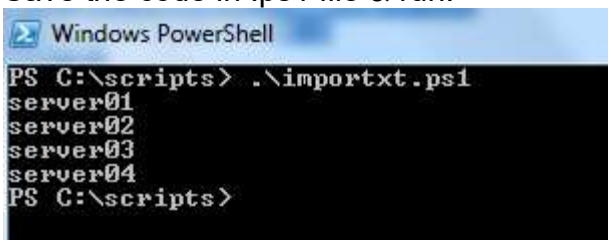
By using below code you can import txt file as an input to your scripts.

```
$servers = Get-content .\servers.txt

$servers | foreach-object {
Write-host $_
}
```

Here are the contents of the servers.txt file



Save the code in .ps1 file & run.

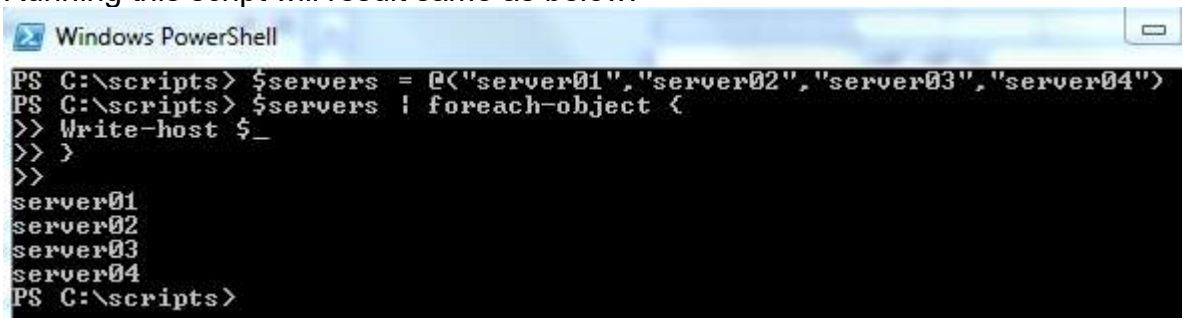


3.3. Input from Array

You can also Input from an array like below.

```
$servers = @("server01","server02","server03","server04")

$servers | foreach-object {
Write-host $_
}
```

Running this script will result same as below:



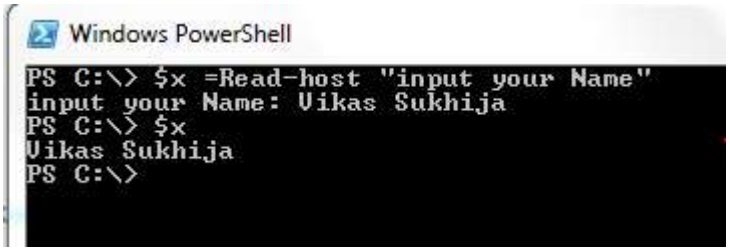
4. Interactive Input

If you want that user should input the values to the script & that too interactively, this section will provide you various such examples.

4.1. Read-host

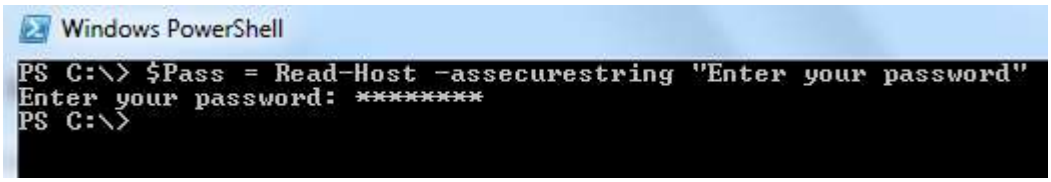
You have used write-host for printing value to the screen, now you can use Read-host to get values that is input by user on the screen.

```
$x =Read-host "input your Name"
```




```
$Pass = Read-Host -assecurestring "Enter your password"
```

-assecurestring will hide the values (this can be used in case you want to enter passwords)



4.2. Parameters

If you want that script should accept parameters, example:

Script.ps1 –firstname “Vikas” –lastname “Sukhija”

```
Param(
  [string]$firstname,
  [string]$lastname
)
```

TIP: Make sure you define these at the beginning of your script.

4.3. GUI Button

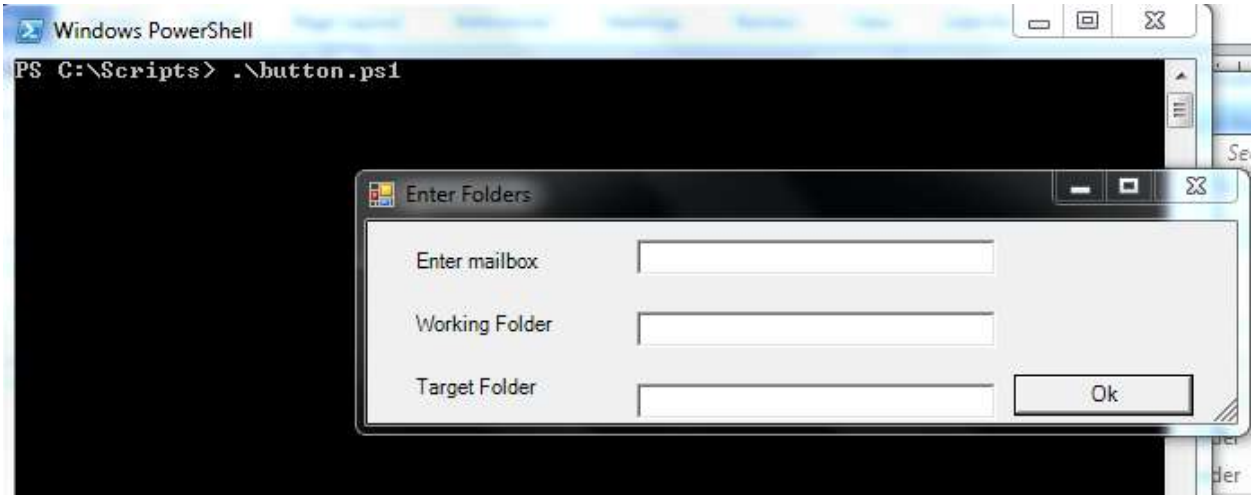
You want to input values in GUI box when executing script than you can use below code.

```
function button ($title,$mailbx, $WF, $TF) {

###################Load Assembly for creating form & button######
[void][System.Reflection.Assembly]::LoadWithPartialName(
"System.Windows.Forms")
[void][System.Reflection.Assembly]::LoadWithPartialName(
"Microsoft.VisualBasic")

#####Define the form size & placement
$form = New-Object "System.Windows.Forms.Form";
$form.Width = 500;
$form.Height = 150;
$form.Text = $title;
$form.StartPosition = [System.Windows.Forms.FormStartPosition]::CenterScreen;

###############Define text label1
$textLabel1 = New-Object "System.Windows.Forms.Label";
$textLabel1.Left = 25;
$textLabel1.Top = 15;

$textLabel1.Text = $mailbx;

###############Define text label2

$textLabel2 = New-Object "System.Windows.Forms.Label";
$textLabel2.Left = 25;
$textLabel2.Top = 50;

$textLabel2.Text = $WF;

###############Define text label3

$textLabel3 = New-Object "System.Windows.Forms.Label";
$textLabel3.Left = 25;
$textLabel3.Top = 85;

$textLabel3.Text = $TF;

############Define text box1 for input
$textBox1 = New-Object "System.Windows.Forms.TextBox";
$textBox1.Left = 150;
$textBox1.Top = 10;
$textBox1.width = 200;

############Define text box2 for input
```

```
$textBox2 = New-Object "System.Windows.Forms.TextBox";
$textBox2.Left = 150;
$textBox2.Top = 50;
$textBox2.width = 200;

############Define text box3 for input

$textBox3 = New-Object "System.Windows.Forms.TextBox";
$textBox3.Left = 150;
$textBox3.Top = 90;
$textBox3.width = 200;

#############Define default values for the input boxes
$defaultValue = ""
$textBox1.Text = $defaultValue;
$textBox2.Text = $defaultValue;
$textBox3.Text = $defaultValue;

#############define OK button
$button = New-Object "System.Windows.Forms.Button";
$button.Left = 360;
$button.Top = 85;
$button.Width = 100;
$button.Text = "Ok";

############# This is when you have to close the form after getting values
$eventHandler = [System.EventHandler]{
$textBox1.Text;
$textBox2.Text;
$textBox3.Text;
$form.Close();};

$button.Add_Click($eventHandler) ;

#############Add controls to all the above objects defined
$form.Controls.Add($button);
$form.Controls.Add($textLabel1);
$form.Controls.Add($textLabel2);
$form.Controls.Add($textLabel3);
$form.Controls.Add($textBox1);
$form.Controls.Add($textBox2);
$form.Controls.Add($textBox3);
$ret = $form.ShowDialog();

#################return values

return $textBox1.Text, $textBox2.Text, $textBox3.Text
}
```

```
$return= button "Enter Folders" "Enter mailbox" "Working Folder" "Target
Folder"
```

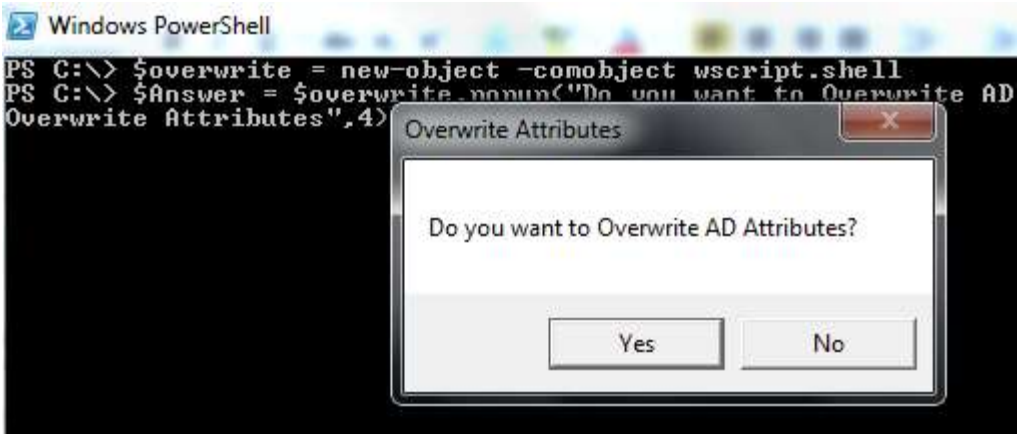
Just copy the above code in the .ps1 & run it, user will see the prompt.

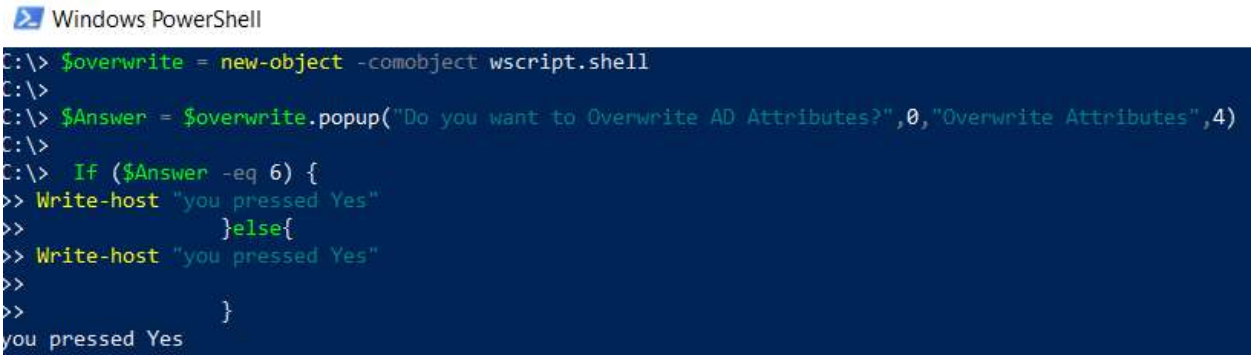


4.4. Prompt

If you want that user should answer Yes or NO.

```
$overwrite = new-object -comobject wscript.shell

$Answer = $overwrite.popup("Do you want to Overwrite AD
Attributes?",0,"Overwrite Attributes",4)

 If ($Answer -eq 6) {
Write-host “you pressed Yes”
   }else{
Write-host “you pressed Yes”

   }
```





5. Adding Snap ins/ Modules

Microsoft & various third-party vendors have built PowerShell Snapins /modules for different products. To use the cmdlets you have to either add those snapins or import those modules in your scripts.

5.1. PowerShell Snapins

If you want ADD exchange snapin than you can use below code.(Example of Exchange 2007 snap in) – Do not know if any one still use exchange 2007 but I have handy example for that as now a days I have seen less use of snapins and more use of modules.

```
If ((Get-PSSnapin | where {$_.Name -match "Exchange.Management"}) -eq $null)
{
    Add-PSSnapin Microsoft.Exchange.Management.PowerShell.Admin
}
```

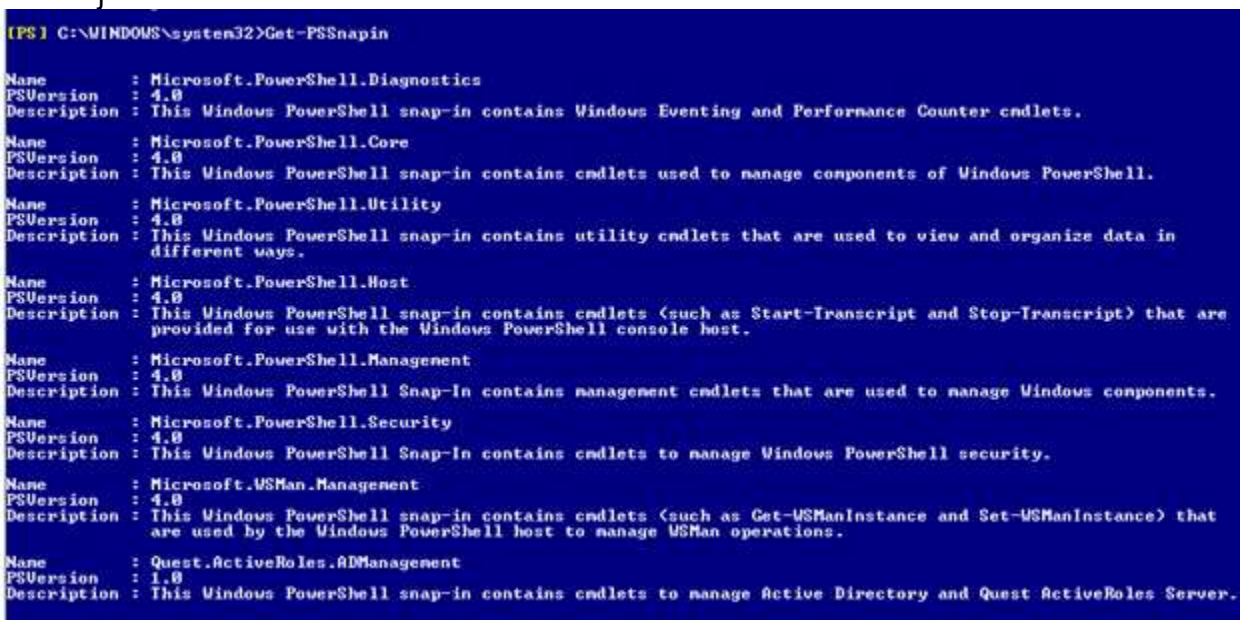
Example of Exchange 2010 Snapin.

```
If ((Get-PSSnapin | Where-Object { $_.Name -match
"Microsoft.Exchange.Management.PowerShell.E2010" }) -eq $null) {
    Add-PSSnapin Microsoft.Exchange.Management.PowerShell.E2010
}
```

You can use **get-pssnapin** in product shell window to check which snapin you want to use.

In below example I want to take advantage of Quest shell, so I used **get-pssnapin** in quest AD shell to know the snapin Name.

```
If ((Get-PSSnapin | where {$_.Name -match "Quest.ActiveRoles"}) -eq $null)
{
    Add-PSSnapin Quest.ActiveRoles.ADManagement
}
```



Note: I am always checking if snapin is already added or not, if its null then only I am adding in current shell.

5.2. Modules

Here is the example on how to import modules

```
If ((Get-Module | where {$_.Name -match "Lync"}) -eq $null)
{
    Import-Module Lync
}
```

Example of importing AzureAD

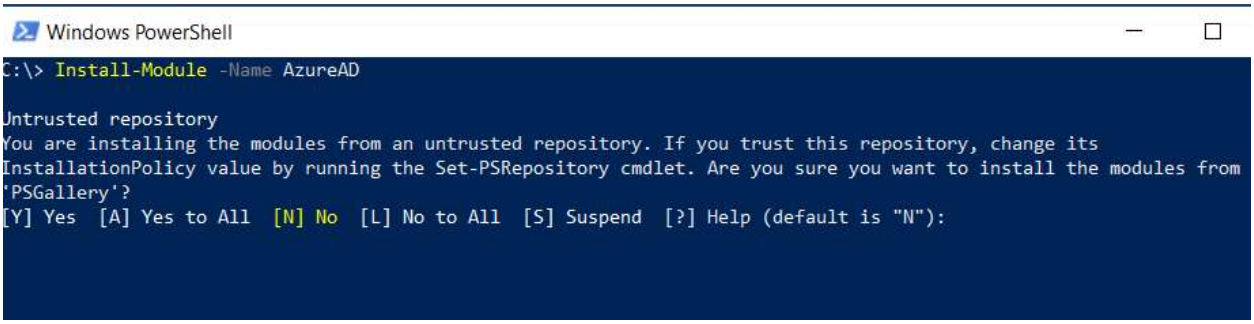
```
If ((Get-Module | where {$_.Name -match "AzureAD"}) -eq $null)
{
   Import-Module AzureAD
}
```

Now a days most of the modules are available in powershell gallery, if you have never heard of it go browse it at https://www.powershellgallery.com/

You can install any module from the gallery using below example commands:

Install-Module -Name AzureAD

Hit yes when you receive below prompt to install the module.



5.3. Import Session

There is another way you can use powershell commands without importing module or adding snapins and this does not require any installation.

You can just import the powershell cmdlets in your session and execute the commands.

Office 365 exchange online and other cloud solution use this approach as well.

You can use this approach to connect to onpremise exchange server as well without installing exchange management shell on your machine.

Example:

```
$Session = New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri
http://Exchnageserver.labtest.com/PowerShell/ -Authentication Kerberos
import-pssession $session -AllowClobber
```

6. Sending Email

Sending email thru scripts is an important aspect of scripting, so here is the code that you can use to send email whenever it is required.

Below code works on all versions of PowerShell

```
$message = new-object Net.Mail.MailMessage
$smtp = new-object Net.Mail.SmtpClient("smtp.lab.com")
$message.From = "donotreply@lab.com"
$message.To.Add("vikas.sukhija@lab.com")

$file =  “c:\file.txt”
$att = new-object Net.Mail.Attachment($file)
$message.IsBodyHtml = $False
$message.Subject = "Enter the subject"
$message.Attachments.Add($att)
$smtp.Send($message)
```

With new versions of powershell there is inbuilt Send-Mailmessage command that you can utilize.

Send-MailMessage -SmtpServer "smtp.lab.com" -To "Vikas.Sukhija@labtest.com" -From "DoNotReply@labtest.com" -Subject "Test Email" -Body "Test Email"

7. Error Reporting

For successful scripting, error reporting is necessary so here are the examples that you can use. You can log errors or send it via email. Errors in PowerShell are stored in inbuilt variable named $error.

7.1. Reporting Error thru Email

Below is the code that can be used to send error via email, you can insert this code in your scripts so that if script resulted in an error it is sent via email.

$error is the default variable in powershell that contains the error if it occurred during execution of the code.

Below we are checking if $error is not equal to null then send the error on email.

After sending error email if we want to clear the error we can use **$error.clear()** so that if we are using iterations error is not sent again if it has not occurred on next iteration.

```
$fromadd = "donotreply@lab.com"
$email1="vikas@lab.com"
$email2="aish@lab.com"
$smtpServer="smtp.lab.com"

if ($error -ne $null)
      {
#SMTP Relay address
$msg = new-object Net.Mail.MailMessage
$smtp = new-object Net.Mail.SmtpClient($smtpServer)
      #Mail sender
$msg.From = $fromadd
#mail recipient
$msg.To.Add($email1)
$msg.To.Add($email2)
$msg.Subject = "DL Modification Script error"
$msg.Body = $error
$smtp.Send($msg)
       }
  else
      {
    Write-host "no errors till now"
      }
```

7.2. Logging Everything including Error

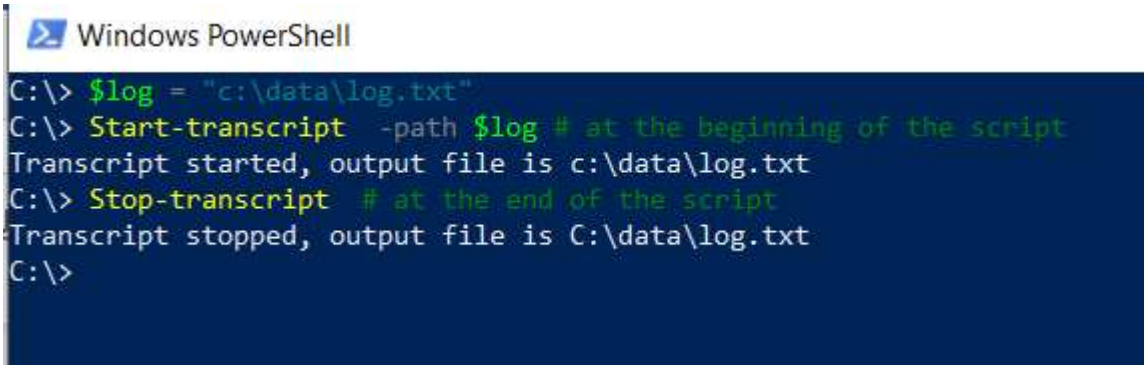
There is an inbuilt PowerShell cmdlet that you can use at the beginning of your script & stop at the end of the script.

```
Start-transcript  # at the beginning of the script
Stop-transcript  # at the end of the script
```

This log will by default get stored in running account My Documents folder.

To store the transcript at different location you can specify the path parameter.

```
$log = “c:\data\log.txt”

Start-transcript  -path $log # at the beginning of the script
Stop-transcript  # at the end of the script
```



7.3. Logging error to Text file

You can also just log errors in Text files, beginning of this Cheat guide we defined logs so we can just utilize those, we can also date stamp the entries, so that we know when the error occurred.

```
Add-Content  $log1 “$date ……. $error”
```

8. Reporting

Many a times You need to create reports csv, excel , HTML etc so this section has codes related to it.

8.1. CSV Report

Export-CSV is the inbuilt cmdlet that can be used to export to csv but in almost all the situations you need a scripting code to format the data in right manner. Here is the example:

```
Collection = @()
Get-Mailbox -ResultSize Unlimited | foreach-object{
$st = get-mailboxstatistics $_.identity
$TotalSize = $st.TotalItemSize.Value.ToMB()
$user = get-user $_.identity
$mbxr = "" | select DisplayName,Alias,RecipientType,TotalItemSizeinMB,
QuotaStatus,
UseDatabaseQuotaDefaults,IssueWarningQuota,ProhibitSendQuota,ProhibitSendRece
iveQuota,
Itemcount, Email,ServerName,Company,Hidden, OrganizationalUnit,
RecipientTypeDetails,UserAccountControl,Exchangeversion

$mbxr.DisplayName = $_.DisplayName
$mbxr.Alias = $_.Alias
$mbxr.RecipientType = $user.RecipientType
$mbxr.TotalItemSizeinMB = $TotalSize
$mbxr.QuotaStatus = $st.StorageLimitStatus
$mbxr.UseDatabaseQuotaDefaults = $_.UseDatabaseQuotaDefaults
$mbxr.IssueWarningQuota = $_.IssueWarningQuota.Value
$mbxr.ProhibitSendQuota = $_.ProhibitSendQuota.Value
$mbxr.ProhibitSendReceiveQuota = $_.ProhibitSendReceiveQuota.Value
$mbxr.Itemcount = $st.Itemcount
$mbxr.Email = $_.PrimarySmtpAddress
$mbxr.ServerName = $st.ServerName
$mbxr.Company = $user.Company
$mbxr.Hidden = $_.HiddenFromAddressListsEnabled
$mbxr.RecipientTypeDetails = $_.RecipientTypeDetails
$mbxr.OrganizationalUnit = $_.OrganizationalUnit
$mbxr.UserAccountControl = $_.UserAccountControl
$mbxr.ExchangeVersion= $_.ExchangeVersion
$Collection += $mbxr
}
 #export the collection to csv , define the $output path accordingly
$Collection | export-csv "c:\data\report.csv"
```

Another important aspect of CSV reporting is to export multi valued attributes. Example for extracting recipients in case of exchange tracking logs

@{Name=”Recipents”;Expression={$_.recipients}}

Example for extracting recipient values from exchange transport logs.

Get-transportserver | Get-MessageTrackingLog -Start “03/09/2015 00:00:00 AM” -End “03/09/2015 11:59:59 PM” – sender “vikas@lab.com” –resultsize unlimited | select Timestamp,clientip,ClientHostname,ServerIp,ServerHostname,sender,EventId,Message Subject, TotalBytes , SourceContext,ConnectorId,Source , InternalMessageId , MessageId ,@{Name=”Recipents”;Expression={$_.recipients}} | export-csv c:\track.csv

8.2. HTML Reporting

It would be wonderful if we can create HTML dashboards with PowerShell ☺, yes its possible by using below HTML code templates in your scripts & utilizing it.

```
################################HTml Report
Content############################
$report = $reportpath
Clear-Content $report
Add-Content $report "<html>"
Add-Content $report "<head>"
Add-Content $report "<meta http-equiv='Content-Type' content='text/html;
charset=iso-8859-1'>"
Add-Content $report '<title>Exchange Status Report</title>'
add-content $report '<STYLE TYPE="text/css">'
add-content $report  "<!--"
add-content $report  "td {"
add-content $report  "font-family: Tahoma;"
add-content $report  "font-size: 11px;"
add-content $report  "border-top: 1px solid #999999;"
add-content $report  "border-right: 1px solid #999999;"
add-content $report  "border-bottom: 1px solid #999999;"
add-content $report  "border-left: 1px solid #999999;"
add-content $report  "padding-top: 0px;"
add-content $report  "padding-right: 0px;"
add-content $report  "padding-bottom: 0px;"
add-content $report  "padding-left: 0px;"
add-content $report  "}"
add-content $report  "body {"
add-content $report  "margin-left: 5px;"
add-content $report  "margin-top: 5px;"
add-content $report  "margin-right: 0px;"
add-content $report  "margin-bottom: 10px;"
add-content $report  ""
add-content $report  "table {"
add-content $report  "border: thin solid #000000;"
```

```
add-content $report  "}"
add-content $report  "-->"
add-content $report  "</style>"
Add-Content $report "</head>"
Add-Content $report "<body>"
add-content $report  "<table width='100%'>"
add-content $report  "<tr bgcolor='Lavender'>"
add-content $report  "<td colspan='7' height='25' align='center'>"
add-content $report  "<font face='tahoma' color='#003399'
size='4'><strong>DAG Active Manager</strong></font>"
add-content $report  "</td>"
add-content $report  "</tr>"
add-content $report  "</table>"
add-content $report  "<table width='100%'>"
Add-Content $report  "<tr bgcolor='IndianRed'>"
Add-Content $report  "<td width='10%' align='center'><B>Identity</B></td>"
Add-Content $report  "<td width='5%'
align='center'><B>PrimaryActiveManager</B></td>"
Add-Content $report  "<td width='20%'
align='center'><B>OperationalMachines</B></td>"
 Add-Content $report "</tr>"
###############################Report
Template##################################
if($dblfb -lt $hrs)
{
            Add-Content $report "<td bgcolor= 'Red' align=center>
<B>$dblfb</B></td>"
}
else
{
            Add-Content $report "<td bgcolor= 'Aquamarine' align=center>
<B>$dblfb</B></td>"
}
Add-Content $report "<td bgcolor= 'GainsBoro' align=center>
<B>$dbrd</B></td>"
Add-Content $report "</tr>"
}
Add-content $report  "</table>"
Add-Content $report "</body>"
Add-Content $report "</html>"
```

See examples in below link: Exchange Health Check and AD Health Check

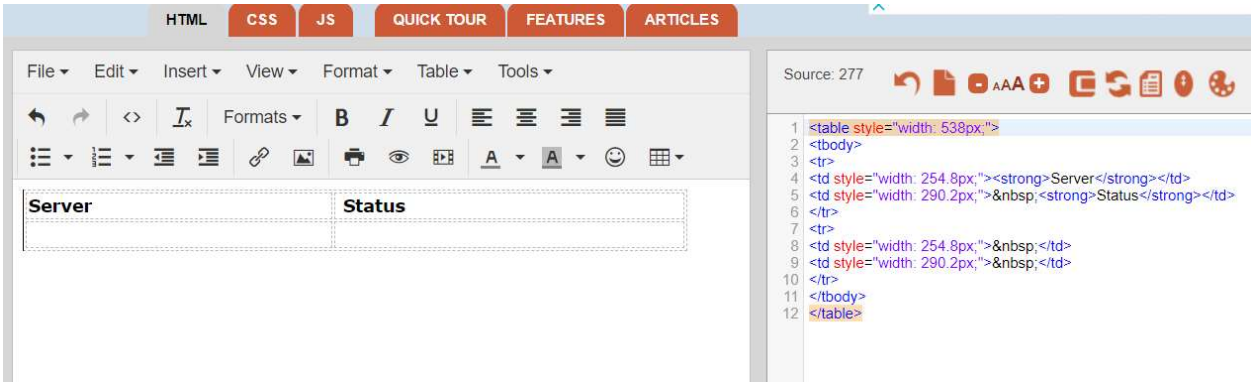
https://techwizard.cloud/exchange-2010-health-check/

https://techwizard.cloud/adhealthcheck/

https://techwizard.cloud/monitor-windows-services-status-remotely/

TIP: You can use HTML online editor to create HTML and use it in your Powershell scripts.

https://html-online.com/editor/



9. Miscellaneous Keywords

These keywords can be used to manipulate the data during parsing inside your scripts.

9.1. Split

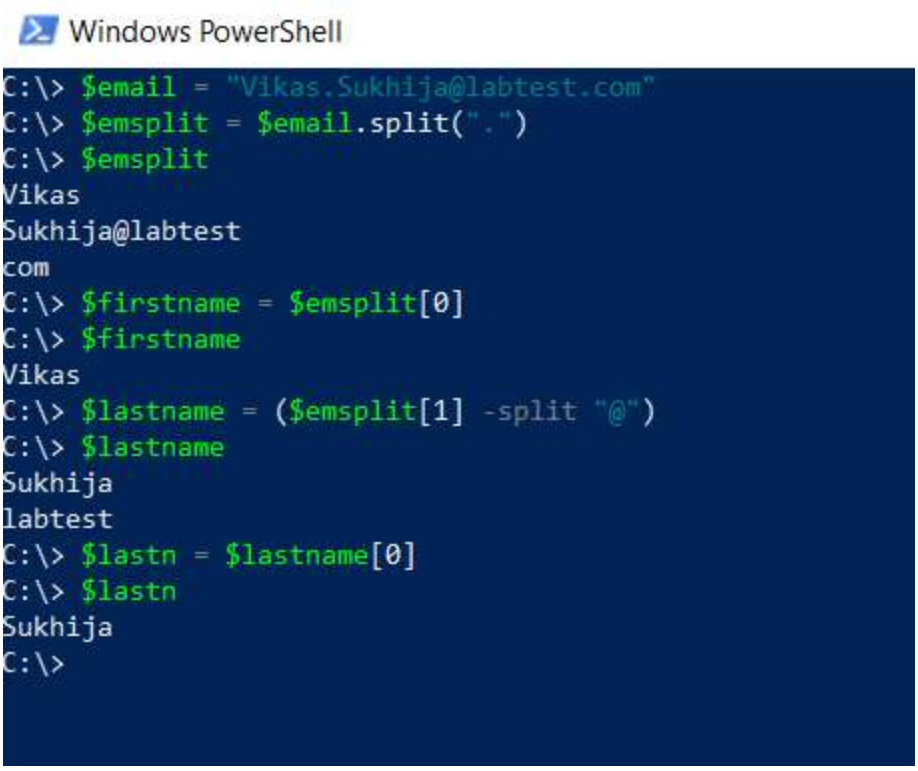
You can use Split, when you want to extract data from string, for example:

Let's extract **First Name** and **Last Name** from it.

We will use split, it can split the string on any character and converts it in array, below we have split the email address on dot ".", after that in element 0 of the array we git the first name but for last name we have to again split at character "@"

```
$email = "Vikas.Sukhija@labtest.com"

$emsplit = $email.split(".")

$firstname = $emsplit[0]

$lastname = ($emsplit[1] -split "@")

$lastn = $lastname[0]
```

See below How I did step by step so that I know the elements of the array and target those. $emsplit[0] and $lastname[0]



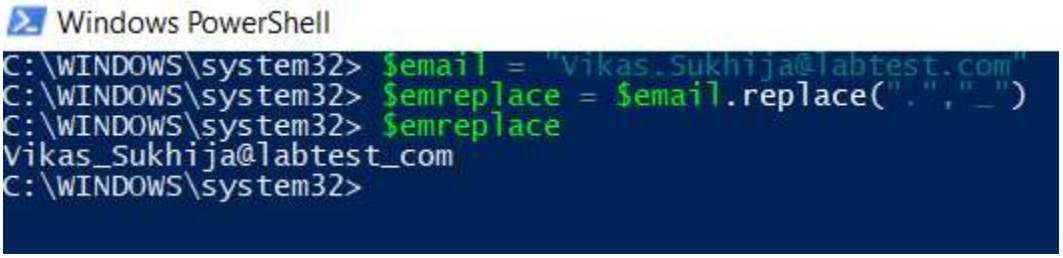
9.2. Replace

You can use replace, when you want to replace data from string, for example:

You want to add underscore instead of dot as you want to update secondary address.

$email = "Vikas.Sukhija@labtest.com"

$emreplace = $email.replace(".","_")



9.3. Select-String

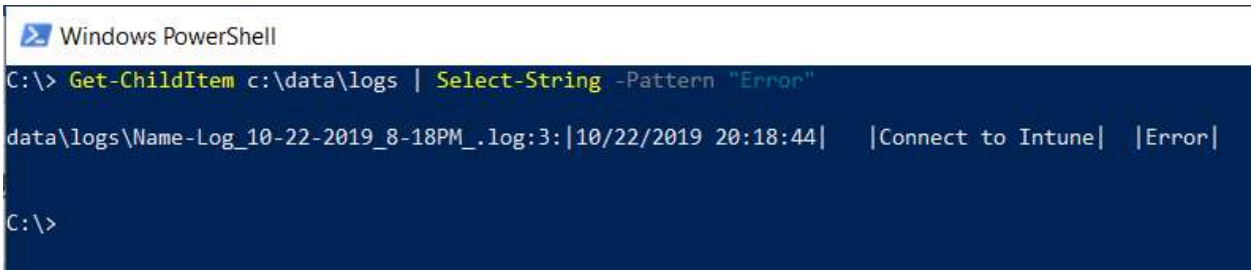
Select-String can do wonders as you can use it to find string inside the files.

For example, if there are large number of files and you want to find a string/keyword exist in particular file.

Get-ChildItem c:\data\logs | Select-String -Pattern “Error"

Here I am searching the string error in all the log files in folder logs.

Only one file has that keyword and I can find exact file using this command.



9.4. Compare-Object

Compare-Object Alias Compare we use many a times to compare two files or two arrays. This is quite fast, I use it many a times by fetching members from a group comparing it with a text file that has user ids to fetch only members that are not already part of the group and add those only instead of processing all members.

Below example add members from one group to other (missing members in group2 but they are in group1)

```
 ############################fetching group1 ##################
$collgroup1 = Get-ADGroup -id "group1" -Properties member |
 Select-Object -ExpandProperty member |
 Get-ADUser |
 Select-Object -ExpandProperty samaccountname

############################fetching group2 ##################
$collgroup2 = Get-ADGroup -id "group2" -Properties member |
 Select-Object -ExpandProperty member |
 Get-ADUser |
 Select-Object -ExpandProperty samaccountname

####################compare two groups####################
$change = Compare-Object -ReferenceObject $collgroup1 -DifferenceObject
$collgroup2

$Addition = $change |
Where-Object -FilterScript {$_.SideIndicator -eq "<="} |
Select-Object -ExpandProperty InputObject

#######adding only members that are missing in group2########
$Addition | ForEach-Object{
     $error.clear()
```

```
      $sam = $_
      Add-ADGroupMember -identity "group2" -Members $sam
 }
```

Similarly, you can do remove operation as well.

There are other nice tricks that you can do with compare-object for example, you have two CSV files, one just has email address of users and others has email address as well as other properties. You want all details from CSV file 2 for users that are in CSV file one.

In below example I did it for Onedrive properties.

```
$importallonedrivesites = import-csv "c:\importonedrives.csv" # onedrive file
with other attributes
$importspofile = import-csv "c:\users.csv" #users email addresses

$change = Compare-Object -ReferenceObject $importallonedrivesites   -
DifferenceObject $importspofile -Property owner -IncludeEqual -PassThru
#owner is the column name for users email addreses

          $change |
          where{$_.SideIndicator -eq "==" -or $_.SideIndicator -eq "=>"} |
          select Owner, Title, url, StorageUsageCurrent, StorageQuota,
StorageQuotaWarningLevel |
          Export-Csv "c:\newfile.csv" -NoTypeInformation
```

10. Product Examples (Daily Use)

Due to my Love for Exchange, I will choose first product as Microsoft Exchange☺.
Please find below Exchange Script excerpts that you can use on day to day basis.

10.1. Microsoft Exchange

10.1.1 Clean Database so that mailboxes appear in disconnected state

```
Get-MailboxServer | Get-MailboxDatabase | Clean-MailboxDatabase
```

10.1.2 Find Disconnected Mailboxes

```
Get-ExchangeServer | Where-Object {$_.IsMailboxServer -eq $true} | ForEach-
Object { Get-MailboxStatistics -Server $_.Name | Where-Object
{$_.DisconnectDate -notlike ''}}
```

10.1.3 Clustered Mailbox Status (2007)

```
Get-ExchangeServer | where {$_.ServerRole -eq "Mailbox" -and $_.Edition -eq
"Enterprise"} | Get-ClusteredMailboxServerStatus
```

10.1.4 Extract Message accept from

```
Get-distributiongroup "dl name" | foreach { $_.AcceptMessagesonlyFrom} | add-
content  "c:/output/abc.txt"
```

10.1.5 Active Sync Stats

```
Get-CASMailbox -ResultSize unlimited | where {$_.ActiveSyncEnabled -eq
"true"} | ForEach-Object {Get-ActiveSyncDeviceStatistics -
Mailbox:$_.identity} | select Devicetype, DeviceID,DeviceUserAgent,
FirstSyncTime, LastSuccessSync, Identity, DeviceModel, DeviceFriendlyName,
DeviceOS | Export-Csv c:\activesync.csv
```

10.1.6 Message Tracking

```
Get-transportserver | Get-MessageTrackingLog -Start "03/09/2015 00:00:00 AM"
-End "03/09/2015 11:59:59 PM" -sender  "vikas@lab.com" -resultsize unlimited
| select
Timestamp,clientip,ClientHostname,ServerIp,ServerHostname,sender,EventId,Mess
ageSubject, TotalBytes , SourceContext,ConnectorId,Source , InternalMessageId
, MessageId ,@{Name="Recipents";Expression={$_.recipients}} | export-csv
c:\track.csv
```

10.1.7 Search mailbox / Delete Messages

Searchonly

```
import-csv c:\tmp\messagesubject.csv | foreach {Search-Mailbox $_.alias -
SearchQuery subject:"Test SUbject" -TargetMailbox "Exmontest" -TargetFolder
"Logs" -LogOnly -LogLevel Full} >c:\tmp\output.txt
```

Delete

```
import-csv c:\tmp\messagesubject.csv | foreach {Search-Mailbox $_.alias -
SearchQuery subject:"Test Schedule" -DeleteContent -force} >c:\tmp\output.txt
```

Delete & log

```
import-csv c:\tmp\messagesubject.csv | foreach {Search-Mailbox $_.alias  -
SearchQuery Subject:"test Story",Received:>'5/23/2018'  -TargetMailbox
"Exmontest" -TargetFolder "Logs"  -deletecontent -force} >c:\tmp\testlog-23-
29-left.txt
```

10.1.8 Exchange Quota Report

This example has been shared under Export-CSV as well.

```
#format Date
$date = get-date -format d
$date = $date.ToString().Replace("/", "-")
$output = ".\" + "QuotaReport_" + $date + "_.csv"

Collection = @()
Get-Mailbox -ResultSize Unlimited | foreach-object{
$st = get-mailboxstatistics $_.identity
$TotalSize = $st.TotalItemSize.Value.ToMB()
$user = get-user $_.identity
$mbxr = "" | select DisplayName,Alias,RecipientType,TotalItemSizeinMB,
QuotaStatus,
```

```
UseDatabaseQuotaDefaults,IssueWarningQuota,ProhibitSendQuota,ProhibitSendRece
iveQuota,
Itemcount, Email,ServerName,Company,Hidden, OrganizationalUnit,
RecipientTypeDetails,UserAccountControl,Exchangeversion

$mbxr.DisplayName = $_.DisplayName
$mbxr.Alias = $_.Alias
$mbxr.RecipientType = $user.RecipientType
$mbxr.TotalItemSizeinMB = $TotalSize
$mbxr.QuotaStatus = $st.StorageLimitStatus
$mbxr.UseDatabaseQuotaDefaults = $_.UseDatabaseQuotaDefaults
$mbxr.IssueWarningQuota = $_.IssueWarningQuota.Value
$mbxr.ProhibitSendQuota = $_.ProhibitSendQuota.Value
$mbxr.ProhibitSendReceiveQuota = $_.ProhibitSendReceiveQuota.Value
$mbxr.Itemcount = $st.Itemcount
$mbxr.Email = $_.PrimarySmtpAddress
$mbxr.ServerName = $st.ServerName
$mbxr.Company = $user.Company
$mbxr.Hidden = $_.HiddenFromAddressListsEnabled
$mbxr.RecipientTypeDetails = $_.RecipientTypeDetails
$mbxr.OrganizationalUnit = $_.OrganizationalUnit
$mbxr.UserAccountControl = $_.UserAccountControl
$mbxr.ExchangeVersion= $_.ExchangeVersion
$Collection += $mbxr
}
 #export the collection to csv , define the $output path accordingly
$Collection | export-csv $output
```

10.1.9 Set Quota

```
#1GB MailBox Limit (Must have the $false included):

set-mailbox testmailbox -UseDatabaseQuotaDefaults $false -IssueWarningQuota
997376KB -ProhibitSendQuota 1048576KB -ProhibitSendReceiveQuota 4194304KB

#2GB MailBox Limit (Must have the $false included):

set-mailbox "testmailbox" -UseDatabaseQuotaDefaults $false -IssueWarningQuota
1.75GB -ProhibitSendQuota 2GB -ProhibitSendReceiveQuota 4GB

#3GB MailBox Limit (Must have the $false included):

set-mailbox "testmailbox" -UseDatabaseQuotaDefaults $false -IssueWarningQuota
2.75GB -ProhibitSendQuota 3GB -ProhibitSendReceiveQuota 5GB
```

10.2. Active Directory

Active directory is Lifeline of every MS product, By using PowerShell you can automate various AD components.

Below are the methods that you can use for Active directory scripting thru PowerShell:

- Active Directory Module
- Quest Management Shell for Active directory
- ADSI

My favorite was Quest management Shell followed by Microsoft Active Directory Module. Quest shell is free & can be downloaded

But as Microsoft has added updates so it is at par or better now than Quest.

One more reason for it is that Quest was acquired by Dell and then they changed the free version to paid, although now again quest is on its own but I have not seen free Quest AD shell although some one is still offering the old version which still works.

Download from below link

https://www.powershelladmin.com/wiki/Quest_ActiveRoles_Management_Shell_Download

ActiveRoles Management Shell for ActiveDirectory


Download Quest ActiveRoles Mangement Shell Version 1.5.1

Here are download links for the x64 and x86 versions of the Quest ActiveRoles AD Management Shell version 1.5.1 (last free version).

NB! Before installing, you will be able to see that the file is signed by Quest, so the files are legit.

They're wrapped in zip files since I already added that file type as an allowed file type to upload. Inside there's an MSI file with the same name (signed by Quest).

- 64-bit version: Quest ActiveRolesManagementShellforActiveDirectoryx64 151.zip
- 32-bit version: Quest ActiveRolesManagementShellforActiveDirectoryx86 151.zip

10.2.1 Export Group members

Just a single line of code will work

Using Quest

```
Get-QADGroupMember "group Name" | select Name, Type | Export-Csv
.\members.csv
```

Using AD Module

```
Get-ADGroup -identity "group Name"  -Properties member | Select-Object -
ExpandProperty member | Get-ADUser -Properties
```

```
DisplayName,Samaccountname,mail,employeeid | export-csv c:\exportgroup.csv -
notypeinfo
```

10.2.2 Set values for Ad attributes

Here is the Example code that can be used to set AD attributes

Using Quest

```
Set-QADUser -identity samaccountname -ObjectAttributes @{extensionattribute10
= "IntuneCommCompleted"}
```

Using AD Module

```
Set-ADUser -identity samaccountanme -replace @{"extensionattribute10" =
"IntuneCommCompleted"}
```

10.2.3 Export Active Directory attributes

This example is for calling excel as well as using quest ☺

```
# call excel for writing the results
$objExcel = new-object -comobject excel.application
$workbook = $objExcel.Workbooks.Add()
$worksheet=$workbook.ActiveSheet
$objExcel.Visible = $False # true or false to set as visible on screen or not
$cells=$worksheet.Cells
# define top level cell
$cells.item(1,1)="UserId"
$cells.item(1,2)="FirstName"
$cells.item(1,3)="LastName"
$cells.item(1,4)="Employeeid"
$cells.item(1,5)="email"
$cells.item(1,6)="Office"
$cells.item(1,7)="Department"
$cells.item(1,8)="Title"
$cells.item(1,9)="Company"
$cells.item(1,10)="City"
$cells.item(1,11)="State"
$cells.item(1,12)="Country"
#intitialize row out of the loop
$row = 2
#import quest management Shell
if ( (Get-PSSnapin -Name Quest.ActiveRoles.ADManagement -ErrorAction
SilentlyContinue) -eq $null )
{
    Add-PsSnapin Quest.ActiveRoles.ADManagement
}
```

```
$data = get-qaduser -IncludedProperties "CO", "extensionattribute1" #-
sizelimit 0
#loop thru users
foreach ($i in $data){
#initialize column within the loop so that it always loop back to column 1
$col = 1
$userid=$i.Name
$FisrtName=$i.givenName
$LastName=$i.sn
$Employeeid=$i.extensionattribute1
$email=$i.PrimarySMTPAddress
$office=$i.Office
$Department=$i.Department
$Title=$i.Title
$Company=$i.Company
$City=$i.l
$state=$i.st
$Country=$i.CO
Write-host "Processing.............................$userid"
$cells.item($row,$col) = $userid
$col++
$cells.item($row,$col) = $FisrtName
$col++
$cells.item($row,$col) = $LastName
$col++
$cells.item($row,$col) = $Employeeid
$col++
$cells.item($row,$col) = $email
$col++
$cells.item($row,$col) = $office
$col++
$cells.item($row,$col) = $Department
$col++
$cells.item($row,$col) = $Title
$col++
$cells.item($row,$col) = $Company
$col++
$cells.item($row,$col) = $City
$col++
$cells.item($row,$col) = $state
$col++
$cells.item($row,$col) = $Country
$col++
$row++}
#formatting excel
$range = $objExcel.Range("A2").CurrentRegion
$range.ColumnWidth = 30
$range.Borders.Color = 0
$range.Borders.Weight = 2
$range.Interior.ColorIndex = 37
```

```
$range.Font.Bold = $false
$range.HorizontalAlignment = 3
# Headings in Bold
$cells.item(1,1).font.bold=$True
$cells.item(1,2).font.bold=$True
$cells.item(1,3).font.bold=$True
$cells.item(1,4).font.bold=$True
$cells.item(1,5).font.bold=$True
$cells.item(1,6).font.bold=$True
$cells.item(1,7).font.bold=$True
$cells.item(1,8).font.bold=$True
$cells.item(1,9).font.bold=$True
$cells.item(1,10).font.bold=$True
$cells.item(1,11).font.bold=$True
$cells.item(1,12).font.bold=$True
#save the excel file
$filepath = "c:\exportAD.xlsx" #save the excel file
$workbook.saveas($filepath)
$workbook.close()
$objExcel.Quit()
```

Same Example using native Active directory Module

```
# call excel for writing the results
$objExcel = new-object -comobject excel.application
$workbook = $objExcel.Workbooks.Add()
$worksheet=$workbook.ActiveSheet
$objExcel.Visible = $True # true or false to set as visible on screen or not
$cells=$worksheet.Cells
# define top level cell
$cells.item(1,1)="UserId"
$cells.item(1,2)="FirstName"
$cells.item(1,3)="LastName"
$cells.item(1,4)="Employeeid"
$cells.item(1,5)="email"
$cells.item(1,6)="Office"
$cells.item(1,7)="Department"
$cells.item(1,8)="Title"
$cells.item(1,9)="Company"
$cells.item(1,10)="City"
$cells.item(1,11)="State"
$cells.item(1,12)="Country"
#intitialize row out of the loop
$row = 2
#import AD management Shell
Import-module Activedirectory
```

```
$data = Get-ADUser -Filter {Enabled -eq $True} -Properties
extensionattribute1,mail,physicalDeliveryOfficeName,Department,title,Company,
l,st,co -ResultSetSize 1000 #define the size
#loop thru users
foreach ($i in $data){
#initialize column within the loop so that it always loop back to column 1
$col = 1
$userid=$i.Name
$FisrtName=$i.givenName
$LastName=$i.surname
$Employeeid=$i.extensionattribute1
$email=$i.mail
$office=$i.physicalDeliveryOfficeName
$Department=$i.Department
$Title=$i.Title
$Company=$i.Company
$City=$i.l
$state=$i.st
$Country=$i.CO
Write-host "Processing.................................$userid"
$cells.item($row,$col) = $userid
$col++
$cells.item($row,$col) = $FisrtName
$col++
$cells.item($row,$col) = $LastName
$col++
$cells.item($row,$col) = $Employeeid
$col++
$cells.item($row,$col) = $email
$col++
$cells.item($row,$col) = $office
$col++
$cells.item($row,$col) = $Department
$col++
$cells.item($row,$col) = $Title
$col++
$cells.item($row,$col) = $Company
$col++
$cells.item($row,$col) = $City
$col++
$cells.item($row,$col) = $state
$col++
$cells.item($row,$col) = $Country
$col++
$row++}
#formatting excel
$range = $objExcel.Range("A2").CurrentRegion
$range.ColumnWidth = 30
$range.Borders.Color = 0
$range.Borders.Weight = 2
```

```
$range.Interior.ColorIndex = 37
$range.Font.Bold = $false
$range.HorizontalAlignment = 3
# Headings in Bold
$cells.item(1,1).font.bold=$True
$cells.item(1,2).font.bold=$True
$cells.item(1,3).font.bold=$True
$cells.item(1,4).font.bold=$True
$cells.item(1,5).font.bold=$True
$cells.item(1,6).font.bold=$True
$cells.item(1,7).font.bold=$True
$cells.item(1,8).font.bold=$True
$cells.item(1,9).font.bold=$True
$cells.item(1,10).font.bold=$True
$cells.item(1,11).font.bold=$True
$cells.item(1,12).font.bold=$True
#save the excel file
$filepath = "c:\exportAD.xlsx" #save the excel file
$workbook.saveas($filepath)
$workbook.close()
$objExcel.Quit()
```

10.2.4 Add members to the group from text file

Using Quest management Shell you can use below syntax

```
$users = Get-Content C:\Users.txt   # samccountnames of users in text file
$groupname = "Group Name"
$users | ForEach-Object{
$user = $_
Write-host "adding $user to $groupname" -foregroundcolor green
Add-QADGroupMember -Identity $groupname -Member $user
}
```

Similarly, in native Active directory module

```
$users = Get-Content C:\Users.txt    # samccountnames of users in text file

$groupname = "Group Name"
$users | ForEach-Object{
$user = $_
Write-host "adding $user to $groupname" -foregroundcolor green
Add-ADGroupMember -id $groupname -members $user
}
```

10.2.5 Remove members to the group from text file

Using Quest management Shell you can use below syntax

```
$users = Get-Content C:\Users.txt   # samccountnames of users in text file
$groupname = "Group Name"
$users | ForEach-Object{
$user = $_
Write-host "adding $user to $groupname" -foregroundcolor green
Remove-QADGroupMember -Identity $groupname -Member $user -confirm:$false
}
```

Similarly, in native Active directory module

```
$users = Get-Content C:\Users.txt    # samccountnames of users in text file

$groupname = "Group Name"
$users | ForEach-Object{
$user = $_
Write-host "adding $user to $groupname" -foregroundcolor green
Remove-ADGroupMember -id $groupname -members $user -confirm:$false
}
```

10.3. Office 365

Office 365 is everywhere so connecting to is important in day to day activities for admins.

You can use the functions that I am sharing and add them to your profile.

Operations:

https://techwizard.cloud/2018/10/25/all-in-one-office-365-powershell-connect-includes-exchange-online-mfa/

https://techwizard.cloud/2016/12/18/all-in-one-office-365-powershell-connect/

- LaunchEOL/RemoveEOL (Exchange Online)
- LaunchEOLMFA/RemoveEOLMFA (Exchange Online with MFA)
- LaunchSOL/RemoveSOL (Skype online)
- LaunchSPO/RemoveSPO (Sharepoint online)
- LaunchCOL/RemoveCOL (Security & Compliance)

- LaunchCOLMFA/RemoveCOLMFA (Security & Compliance)
- LaunchMSOL/RemoveMSOL (MSonline Azure activedirectory)

```
##############################Exchange Online#################
Function LaunchEOL
{
  $UserCredential = Get-Credential

  $Session = New-PSSession -ConfigurationName Microsoft.Exchange -
ConnectionUri https://outlook.office365.com/powershell-liveid/ -Credential
$UserCredential -Authentication Basic -AllowRedirection

  Import-PSSession $Session -Prefix "EOL" -AllowClobber
}



Function RemoveEOL
{
  $Session = Get-PSSession | where {$_.ComputerName -like
"outlook.office365.com"}
  Remove-PSSession $Session
}
##############################Exchange MFA Online#################
Function LaunchEOLMFA
{
  Import-Module $((Get-ChildItem -Path $($env:LOCALAPPDATA+"\Apps\2.0\") -
Filter Microsoft.Exchange.Management.ExoPowershellModule.dll -Recurse
).FullName|
    ?{$_ -notmatch "_none_"}|
  select -First 1)
  $EOLSession = New-ExoPSSession
  Import-PSSession $EOLSession -Prefix "EOL" -AllowClobber -Verbose
}

Function RemoveEOLMFA
{
  $Session = Get-PSSession | where {$_.ComputerName -like
"outlook.office365.com"}
  Remove-PSSession $Session
}
########################Skype Online#############################
function LaunchSOL
{
  param
  (
    $Domain,
    $UserCredential
  )
```

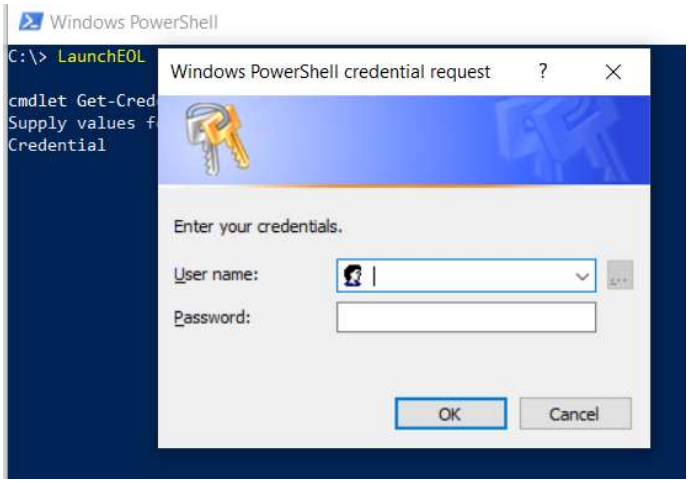
```
  Write-Host "Enter Skype Online Credentials" -ForegroundColor Green
  $CSSession = New-CsOnlineSession -Credential $UserCredential -
OverrideAdminDomain $Domain -Verbose
  Import-PSSession $CSSession -Prefix "SOL" -AllowClobber
}

Function RemoveSOL
{
  $Session = Get-PSSession | where { $_.ComputerName -like
"*.online.lync.com" }
  Remove-PSSession $Session
}

#####################Sharepoint Online#################################

function LaunchSPO
{
  param
  (
    $orgName
  )

  Write-Host "Enter Sharepoint Online Credentials" -ForegroundColor Green
  $UserCredential = Get-Credential
  Connect-SPOService -Url https://$orgName-admin.sharepoint.com -Credential
$UserCredential
}

Function RemoveSPO
{Disconnect-SPOService}

########################Secuirty and Compliance############################

Function LaunchCOL
{
  $UserCredential = Get-Credential

  $Session = New-PSSession -ConfigurationName Microsoft.Exchange -
ConnectionUri https://ps.compliance.protection.outlook.com/powershell-liveid/
-Credential $UserCredential -Authentication Basic -AllowRedirection

  Import-PSSession $Session -Prefix "COL" -AllowClobber
}



Function RemoveCOL
{
```

```
  $Session = Get-PSSession | where {$_.ComputerName -like
"*compliance.protection.outlook.com"}
  Remove-PSSession $Session
}
############################Compliance Online#################
Function LaunchCOLMFA
{
  Import-Module $((Get-ChildItem -Path $($env:LOCALAPPDATA+"\Apps\2.0\") -
Filter Microsoft.Exchange.Management.ExoPowershellModule.dll -Recurse
).FullName|
    ?{$_ -notmatch "_none_"}|
  select -First 1)
  $COLSession = New-EXOPSSession -ConnectionUri
'https://ps.compliance.protection.outlook.com/PowerShell-LiveId'
  Import-PSSession $COLSession -Prefix "COL" -AllowClobber -Verbose
}

Function RemoveCOLMFA
{
  $Session = Get-PSSession | where {$_.ComputerName -like
"*compliance.protection.outlook.com"}
  Remove-PSSession $Session
}
##############################Msonline#########################
function LaunchMSOL
{
  Import-Module msonline
  Write-Host "Enter MS Online Credentials" -ForegroundColor Green
  Connect-MsolService
}

Function RemoveMSOL
{Write-Host "Close Powershell Window - No disconnect available" -
ForegroundColor yellow}
###############################################################
```

10.3.1 Exchange Online Mailbox Report

Now use the above function to launch Exchange online Shell.

In the Powershell → type LaunchEOL and supply Exchange online admin userid/password.


Windows PowerShell
C:\> LaunchEOL
cmdlet Get-Cred
Supply values f
Credential
Windows PowerShell credential request
Enter your credentials.
User name:
Password:
OK
Cancel

Ones you are connected to Exchange online successfully, run below command to extract mailboxes report from office 365.


```
Get-EOLMailbox -ResultSize unlimited | Select
Name,RecipientTypeDetails,PrimarySMTPAddress,UserPrincipalName,litigationhold
enabled,LitigationHoldDuration,PersistedCapabilities,RetentionHoldEnabled,Ret
entionPolicy,RetainDeletedItemsFor,ArchiveName,Archivestatus,ProhibitSendQuot
a,ProhibitSendReceiveQuota,MaxSendSize,MaxReceiveSize,AuditEnabled | export-
csv c:\auditmbx.csv -notypeinfo
```

10.3.2 Exchange Online Message Tracking

In exchange online, extracting the Message tracking is not same as Exchange OnPremise as if results are more in number then it can not be extracted using resultsize unlimited parameter, below is the small script that will do the trick.

```
$index = 1
while ($index -le 1001)
{
Get-EOLMessageTrace -SenderAddress "VikasS@techWizard.cloud" -StartDate
09/20/2019 -EndDate 09/25/2019 -PageSize 5000 -Page $index | export-csv
c:\messagetracking.csv -Append
$index ++
sleep 5
}
```

10.3.3 Searching Unified Log

Office 365 uses unified audit logging and you can audit all the activities using Exchange online Shell.(whether its Sharepoint online or teams or any other product within office 365)

Here is the link for more details:

https://docs.microsoft.com/en-us/microsoft-365/compliance/search-the-audit-log-in-security-and-compliance

Example of extracting Microsoft Teams activity:

```
Search-EOLUnifiedAuditLog -StartDate 1/8/2019 -EndDate 4/7/2019 -RecordType
MicrosoftTeams -UserIds VikasS@sycloudpro.com  -ResultSize:5000 |export-csv
c:\VikasS.csv -notypeinfo
```

Example of extracting Exchange mailbox audit activity:

```
Search-EOLUnifiedAuditLog -StartDate 10/24/2019 -EndDate 10/25/2019 -UserIds
VikasS@syscloudpro.com -recordtype
"ExchangeItemGroup","ExchangeItem","ExchangeAggregatedOperation" -
ResultSize:5000 |export-csv c:\VikasS.csv -notypeinfo
```

Example of Adding or Removing Role member:

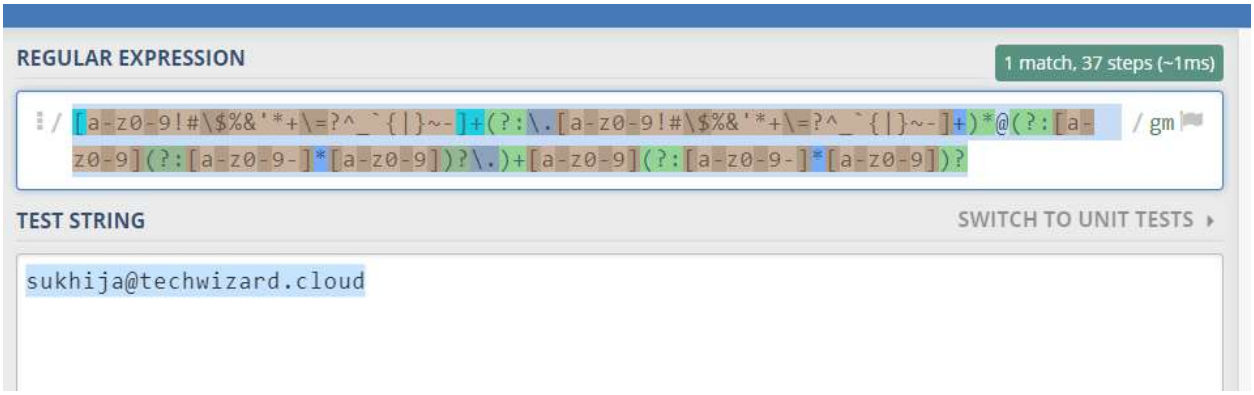
```
Search-EOLUnifiedAuditLog -StartDate 4/16/2019 -EndDate 7/15/2019 -UserIds
VikasS@syscloudpro.com -operations "Add role member to role" -ResultSize:5000
|export-csv c:\VikasS.csv -notypeinfo
```

11. Appendix

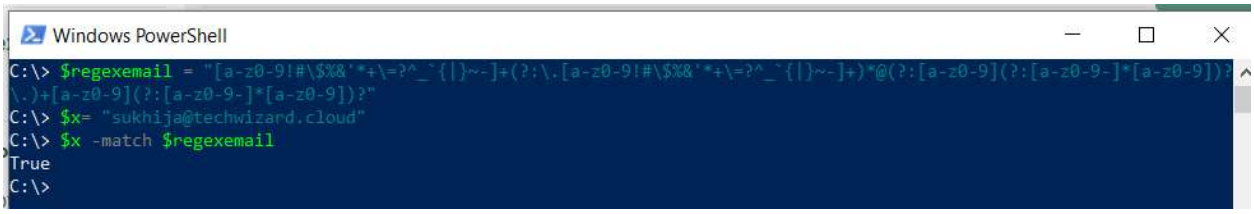
TIP: You can use https://regex101.com/ site to test any regex before using it.

Regular Expressions

```
$regexemail = "[a-z0-9!#\$%&'*+\=?^_`{|}~-]+(?:\.[a-z0-9!#\$%&'*+\=?^_`{|}~-
]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?"

$regexIP = "^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$"
```



This is how we will use it in Powershell, these will be used mainly with match operators.



Sno.	RegexCheat	Comments
1	Reciept_[0-9][0-9][0-9][0-9][0-9][0-9][0-9]\.doc	Contains Reciept_7didgit number.doc
2	(Tickets issued to)(.*)(for travel)	Tickets issued to Vikas Sukhija for travel
3	(.*)Aborted_payment_(.*)	Tell Aborted_payment_(Y075958)
4	(.*)(\([A-Z][0-9][0-9][0-9][0-9][0-9][0-9]\))	(Y782714)
5	(.*)[0-9]{2}[A-Z]{1}[0-9]{6}	Critical_alert_-_36B881478
6	(?<=G0)(.*)(?=a)	G01234a

Excel

```
# ------------------------------------------------------------------------
------------------------
function Release-Ref ($ref) {
([System.Runtime.InteropServices.Marshal]::ReleaseComObject(
[System.__ComObject]$ref) -gt 0)
[System.GC]::Collect()
[System.GC]::WaitForPendingFinalizers()
}
# ------------------------------------------------------------------------
-------------------------
$objExcel = new-object -comobject excel.application
$objExcel.Visible = $False
$objWorkbook = $objExcel.Workbooks.Open($path)
$objWorksheet = $objWorkbook.Worksheets.Item(1)
$Name = $objWorksheet.Cells.Item($intRow, 1).Value()
$filepath = "c:\scripts\Hierarchy.xlsx" ###define path

$workbook.saveas($filepath)
$workbook.close()
$objExcel.Quit()
#Release all the objects used above
$a = Release-Ref($objWorksheet)
$a = Release-Ref($objWorkbook)
$a = Release-Ref($objExcel)
```

Formatting Excel code

```
$range = $objExcel.Range("A2").CurrentRegion
$range.ColumnWidth = 30
$range.Borders.Color = 0
$range.Borders.Weight = 2
$range.Interior.ColorIndex = 37
$range.Font.Bold = $false
$range.HorizontalAlignment = 3
# Headings in Bold
$cells.item(1,1).font.bold=$True
```

Registry

```
[Microsoft.Win32.RegistryValueKind]::Binary  --> REG_BINARY
[Microsoft.Win32.RegistryValueKind]::Qword   --> REG_QWORD
[Microsoft.Win32.RegistryValueKind]::MultiString  --> REG_MULTI_SZ
[Microsoft.Win32.RegistryValueKind]::ExpandString --> REG_EXPAND_SZ
[Microsoft.Win32.RegistryValueKind]::String  --> REG_SZ
-------------------------------------------------------------------------------
----------------
$main = "Localmachine"
$Path = "System\CurrentControlSet\Services\Disk"
$key = "TimeOutValue"
$tout = "60"  #####modified Value of dword #############
-------------------------------------------------------------------------------
-------------------
$reg = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey($main, $Server)
$regKey= $reg.OpenSubKey($path,$True)
$Value = $regkey.GetValue($key)
$regkey.SetValue($key,$tout,[Microsoft.Win32.RegistryValueKind]::DWORD)
```

ADSI

```
Param(
  [string]$user
)
function checkuserDN ($usersm)
{
 $Search = New-Object DirectoryServices.DirectorySearcher([ADSI]"")
$Search.filter =
"(&(objectCategory=user)(objectClass=user)(sAMAccountName=$usersm))"
$findusr=$Search.Findall()
               if ($findusr.count -gt 1)
      {
            $count = 0
 foreach($i in $findusr)
            {
                  write-host $count ": " $i.path
```

```
                  $count = $count + 1
      write-host "multiple matches found"
            }
exit
      }
      elseif ($findusr.count -gt 0)
      {
            return $findusr[0].path
      }
      else
      {
      write-host "no match found"
 }
}
if($user -like $null)
{
"Pls use script as - chkusrdn.ps1 usersamacountname"
}
else
{
checkuserDN $user
}
```

Remove Header Line from CSV

Method1

```
Get-Content .\abc.csv | select -skip 1 | Set-Content .\abc1.csv
```

Method2

```
$a = import .\abc.csv
$a |ForEach-Object{
  $Con_string = $null
  $Con_string = $_.ID, $_.GrpName -join ','
  Write-Host $Con_string
  Add-Content .\abc6.csv $Con_string
}
```

Method3 (avoid CRLF)

```
$text = [System.IO.File]::ReadAllText("$pwd\file.csv") -replace
'^[^\r\n]*\r?\n'
[System.IO.File]::WriteAllText("$pwd\newFile.csv", $text)
```

Method4 (avoid CRLF)

```
$file = Get-Item .\example_test.csv
$reader = $file.OpenText()
# discard the first line
$null = $reader.ReadLine()
# Write the rest of the text to the new file
[System.IO.File]::WriteAllText("$pwd\newFile.csv", $reader.ReadToEnd())
$reader.Close()
```

Add Header Line to Text File

For example you have list of employeeids in text file

14562
67578
65888

```
$filep = "c:\file.txt"

$getNetworkID = Get-Content $filep | where { $_ -ne "" }
@("Employeeid") +  $getNetworkID | Set-Content  $filep -Force #add
emplyeeidheader
```

This cheat book is being constantly updated with new examples and tricks.

You can check my work at http://TechWizard.cloud and browse my scripts at

TechNet Gallery

https://gallery.technet.microsoft.com/scriptcenter/site/search?f%5B0%5D.Type=User&f%5B0%5D.Value=The%20Tech%20Wizard

All of my Technet Scripts/Solution are being migrated to GITHUB now at below url:

https://techwizard.cloud/downloads/