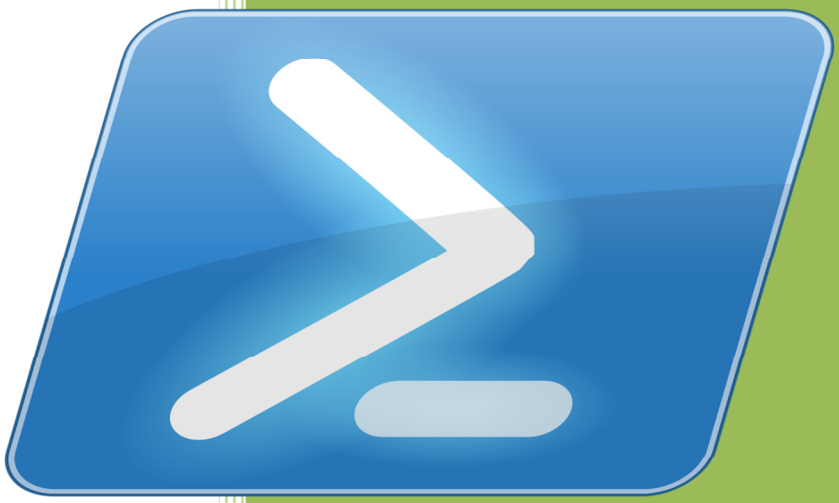
2015

PowerShell Cheat Sheet



Sukhija, Vikas

http://msexchange.me

2/2/2015

POWERSHELL SCRIPTING

FOR

NON SCRIPTERS AND SCRIPTERS

This small ebook 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 ebook is not for fresher's but for experienced IT administrators that want to use scripting & do IT automations. Just copy paste the code blocks from the guide & make complex scripts ☺

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

Author: Vikas Sukhija (MVP 2015) **Website:** http://msexchange.me

Table of Contents

1. PowerShell Basics

Let us start with basic stuff & touch base quickly on Variables, Loops, if than else, Switch and functions. These all stuff will assist you in creating complex scripts.

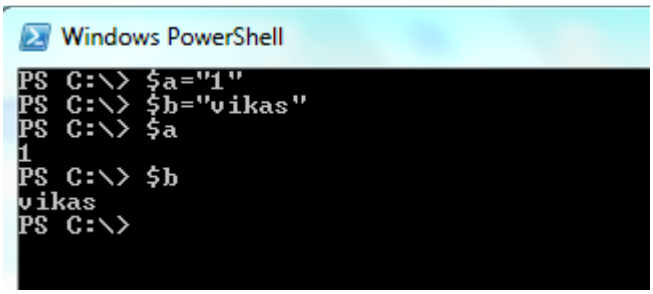
1.1. Variables & Printing

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

PS C:\> Write-host $a

1

PS C:\> Write-host $b

vikas

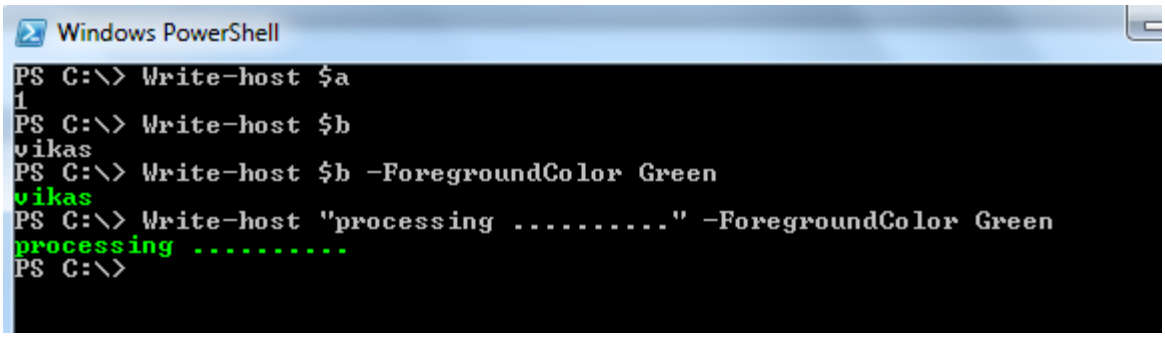
PS C:\> Write-host $b -ForegroundColor Green

vikas

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

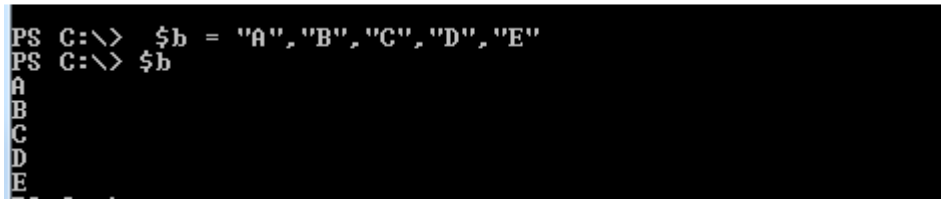
processing .....

PS C:\>



I am not illustrating arrays separately in this ebook as in PowerShell arrays can also be defined in the same way as variables. Below are the examples:

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



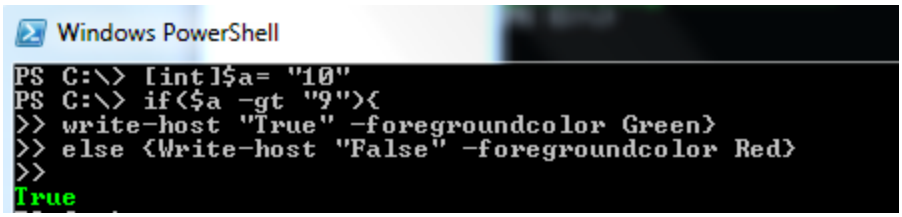
$c= @("server1","server2")

1.2. If Else/ switch

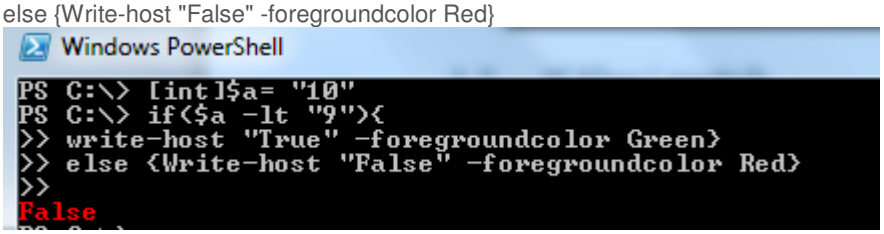
IF ELSE is a condition based processing, it's the bases of any scripting language. If some condition is true you need to process something otherwise process some other thing.

Here I have illustrated two examples; first we defined variable value as 10, than 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.

```
[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.

-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

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

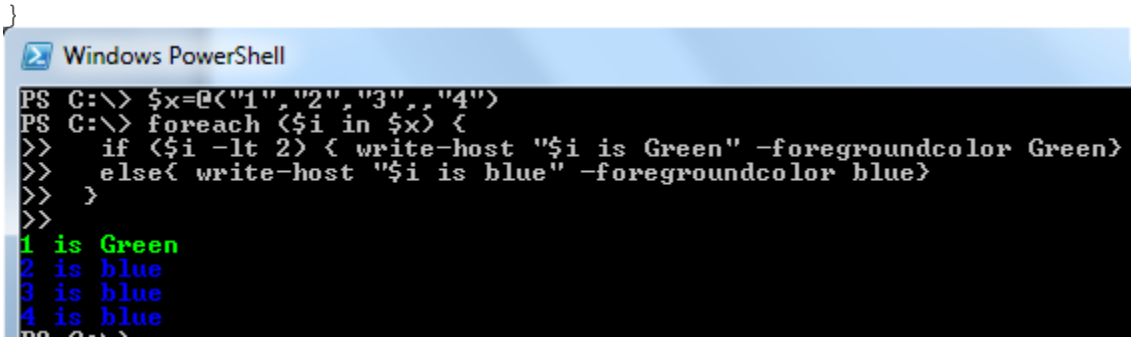
There are three different iterations of For loops:

ForEach
ForEach-Object
For

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

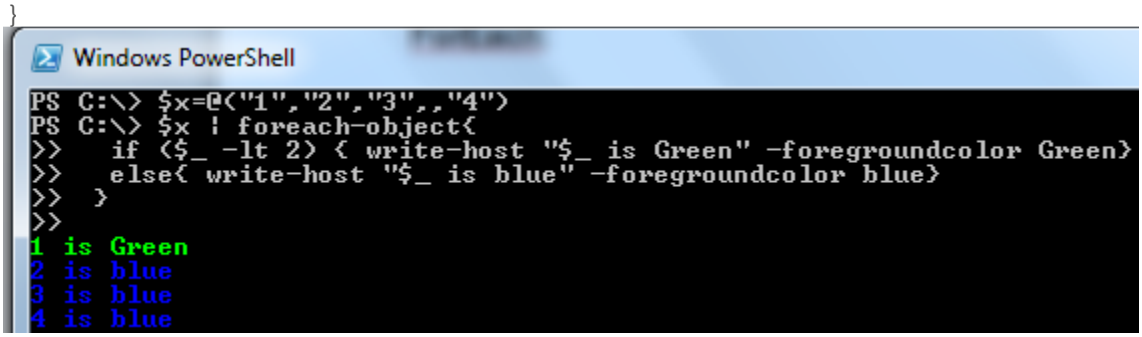
ForEach:

```
$x=@("1","2","3",,"4")
foreach ($i in $x) {
  if ($i -lt 2) { write-host "$i is Green" -foregroundcolor Green}
  else{ write-host "$i is blue" -foregroundcolor blue}
}
```



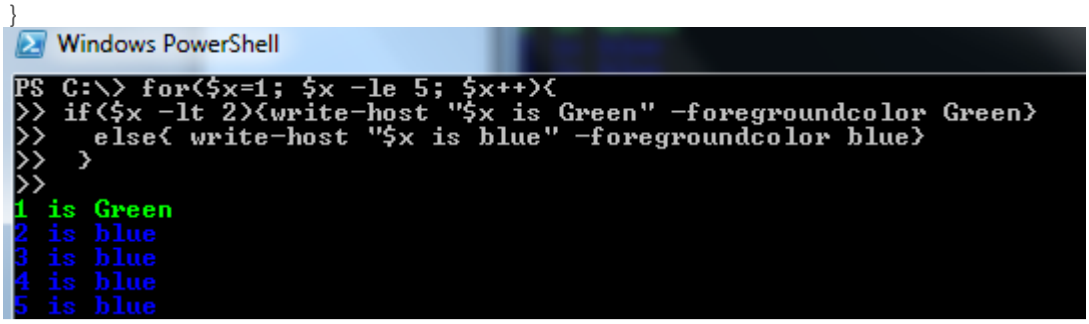
ForEach-Object:

```
$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:

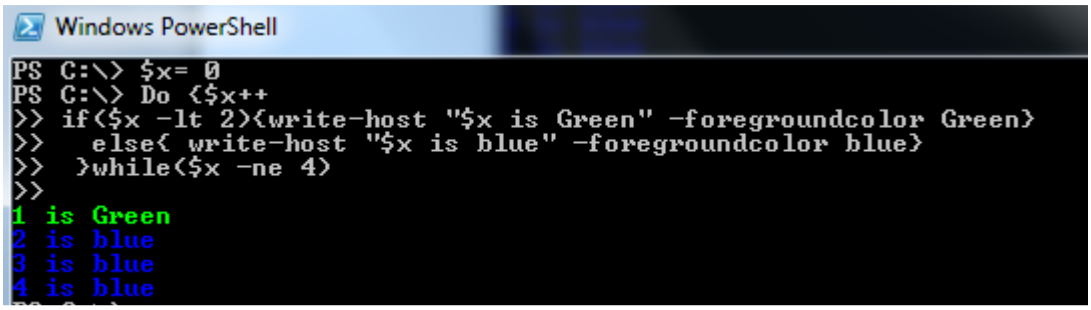
```
for($x=1; $x -le 5; $x++){
if($x -lt 2){write-host "$x is Green" -foregroundcolor Green}
  else{ write-host "$x is blue" -foregroundcolor blue}
}
```



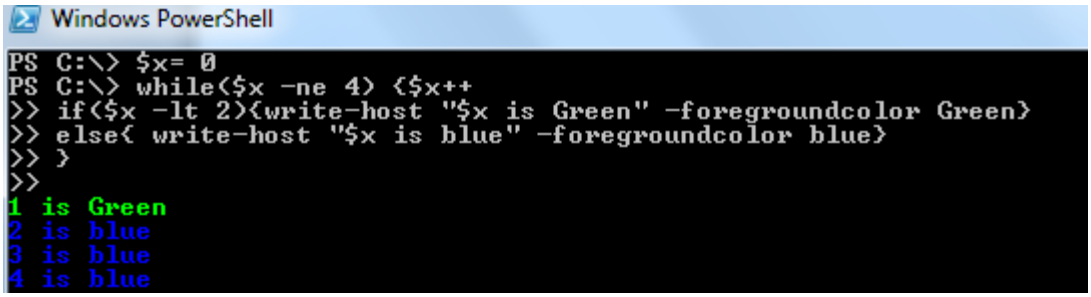
1.3.2 While –Loop

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:

```
$x= 0
Do {$x++
if($x -lt 2){write-host "$x is Green" -foregroundcolor Green}
  else{ write-host "$x is blue" -foregroundcolor blue}
 }while($x -ne 4)
```




```
$x= 0
while($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

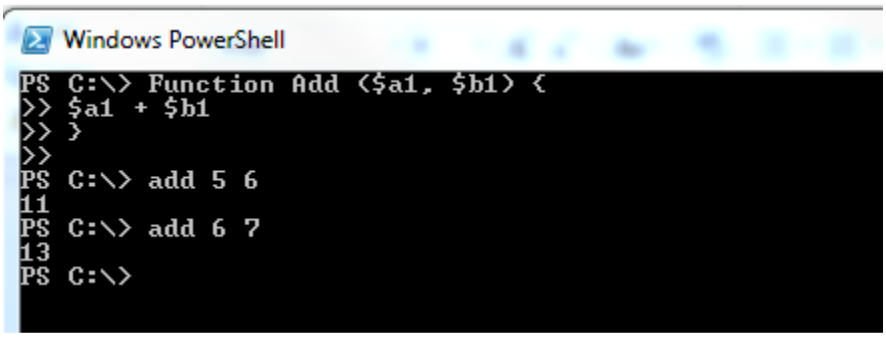
Functions are those entities, ones defined you can call them again & again in any part of the script & you can avoid repetitive code.

Create Function

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

Call function

```
Add 5 6
```



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.

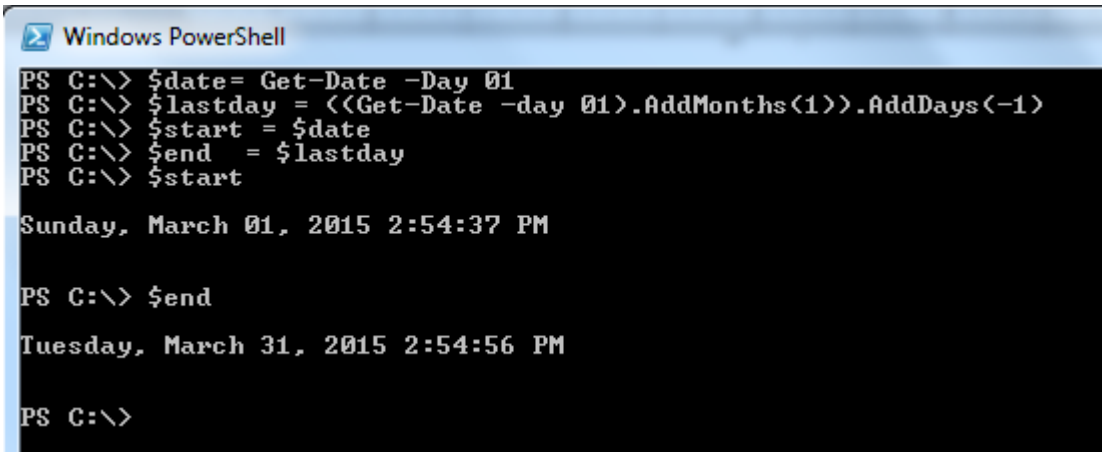
```
$date = get-date -format d
$date = $date.ToString().Replace("/", "-")

$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.

2.2. First day & Last day of Month

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

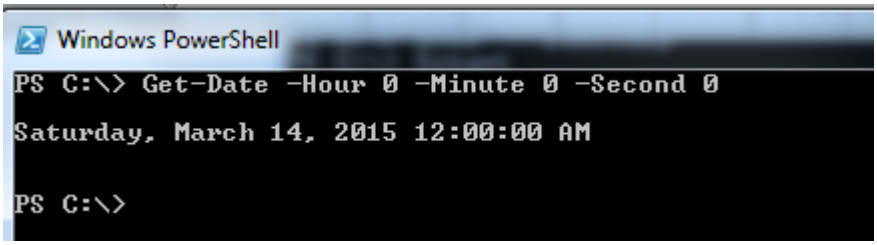
```
$date= Get-Date -Day 01
$lastday = ((Get-Date -day 01).AddMonths(1)).AddDays(-1)

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



2.3. Midnight

To get the midnight

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



3. Input to your scripts

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

3.1. Import CSV

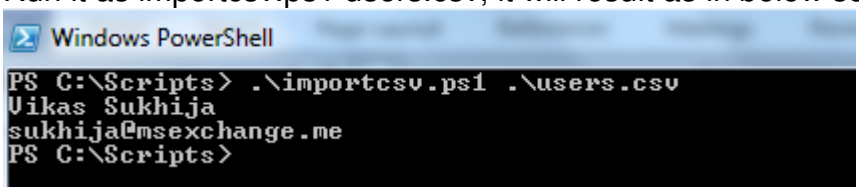
By using below code you can import csv file as an input to your scripts.

```
$data = import-csv $args[0]    #this means that you will specify csv when executing script
foreach ($i in $data) {
Write-host  $i.user
Write-host  $i.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.

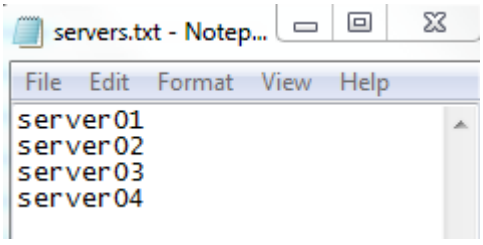


3.2. Import from text file

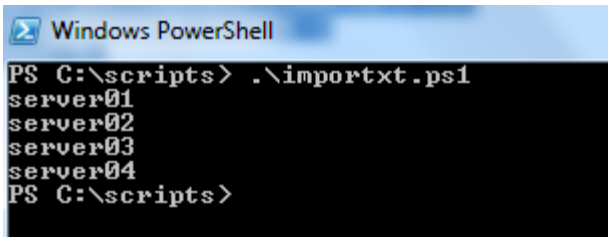
By using below code you can import txt file as an input to your scripts.

```
$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.

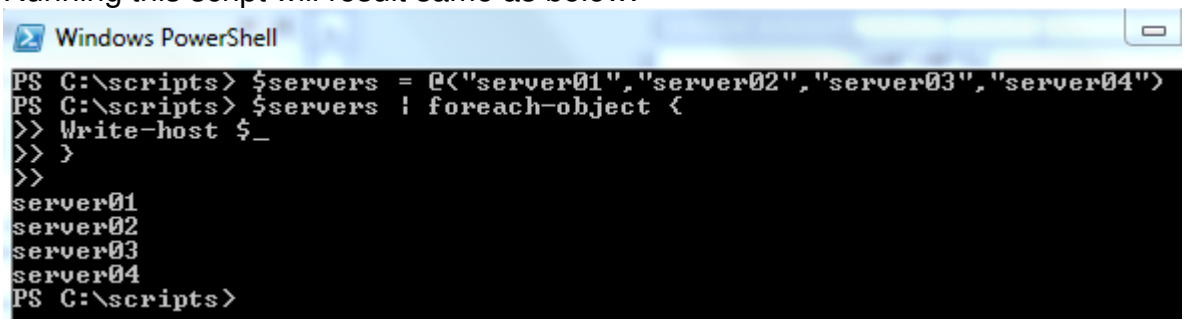


3.3. Input from Array

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:



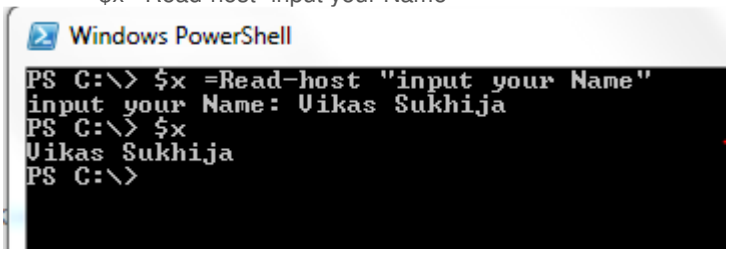
4. Interactive Input

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

4.2. Parameters

If you want that script should accept parameters, example:

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

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

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

4.3. GUI Button

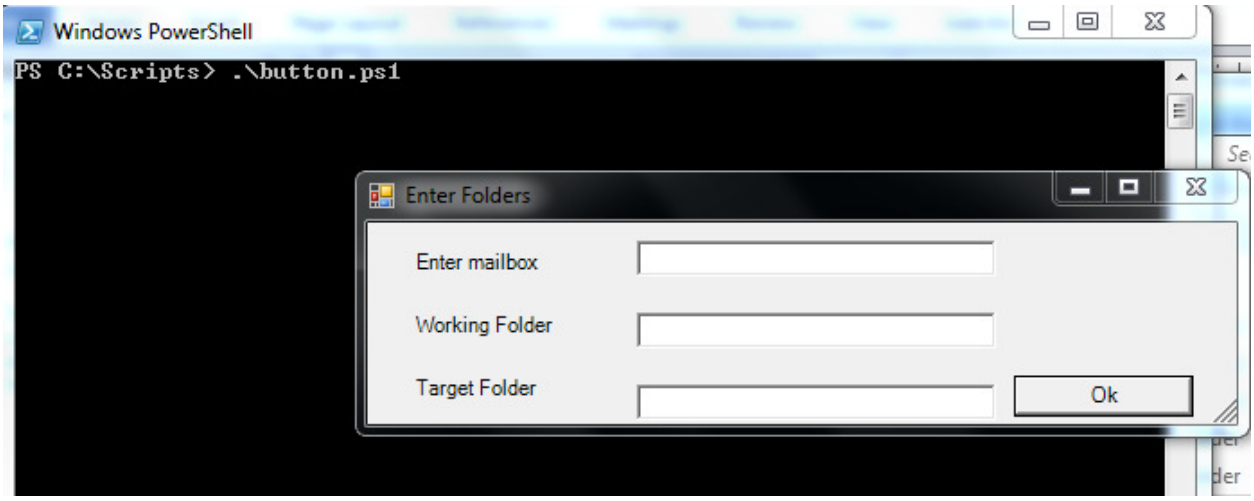
You want to input values in GUI box when executing script than you can use below code.

```
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;
$textBox23.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"
```

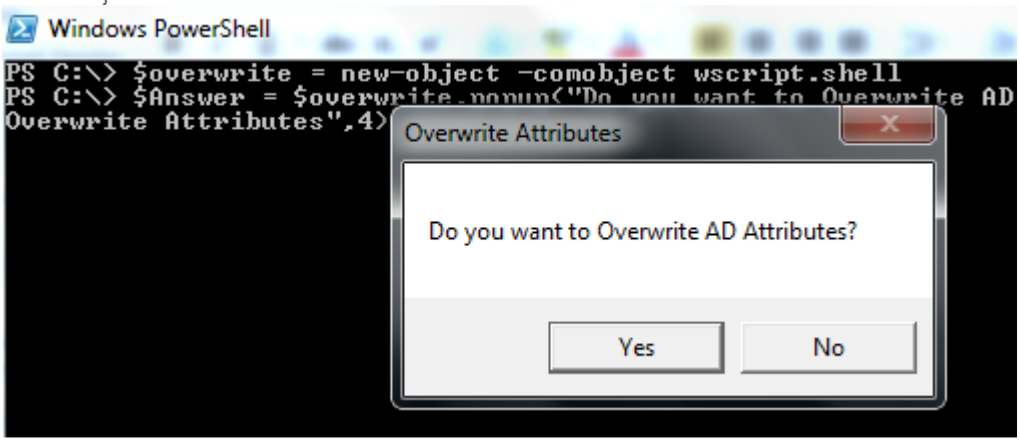
Just copy the above code in the .ps1 & run it, user will see the prompt.



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) {
}else{
}
```



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.

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

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
}
```

```
[PS] C:\WINDOWS\system32>Get-PSSnapin

Name        : Microsoft.PowerShell.Diagnostics
PSVersion   : 4.0
Description : This Windows PowerShell snap-in contains Windows Eventing and Performance Counter cmdlets.

Name        : Microsoft.PowerShell.Core
PSVersion   : 4.0
Description : This Windows PowerShell snap-in contains cmdlets used to manage components of Windows PowerShell.

Name        : Microsoft.PowerShell.Utility
PSVersion   : 4.0
Description : This Windows PowerShell snap-in contains utility cmdlets that are used to view and organize data in
              different ways.

Name        : Microsoft.PowerShell.Host
PSVersion   : 4.0
Description : This Windows PowerShell snap-in contains cmdlets (such as Start-Transcript and Stop-Transcript) that are
              provided for use with the Windows PowerShell console host.

Name        : Microsoft.PowerShell.Management
PSVersion   : 4.0
Description : This Windows PowerShell Snap-In contains management cmdlets that are used to manage Windows components.

Name        : Microsoft.PowerShell.Security
PSVersion   : 4.0
Description : This Windows PowerShell Snap-In contains cmdlets to manage Windows PowerShell security.

Name        : Microsoft.WSMan.Management
PSVersion   : 4.0
Description : This Windows PowerShell snap-in contains cmdlets (such as Get-WSManInstance and Set-WSManInstance) that
              are used by the Windows PowerShell host to manage WSMan operations.

Name        : Quest.ActiveRoles.ADManagement
PSVersion   : 1.0
Description : This Windows PowerShell snap-in contains cmdlets to manage Active Directory and Quest ActiveRoles Server.
```

5.2. Modules

Here is the example on how to import modules

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

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.

```
$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 = $output3  ###please define the file path in variable $output3  ###
$att = new-object Net.Mail.Attachment($file)
$message.IsBodyHtml = $False
$message.Subject = "Enter the subject"
$message.Attachments.Add($att)
$smtp.Send($message)
```

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.

```
$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.

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,ProhibitSendReceiveQuota,
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
```

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}}

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,MessageSubject, 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://gallery.technet.microsoft.com/scriptcenter/Exchange-2010-Health-Check-7be55c87

https://gallery.technet.microsoft.com/scriptcenter/Active-Directory-Health-709336cd

9. Product Examples

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

9.1. Microsoft Exchange

9.1.1 Clean Database so that mailboxes appear in disconnected state

Get-MailboxServer | Get-MailboxDatabase | Clean-MailboxDatabase

9.1.2 Find Disconnected Mailboxes

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

9.1.3 Clustered Mailbox Status (2007)

Get-ExchangeServer | where {$_.ServerRole -eq “Mailbox” -and $_.Edition –eq “Enterprise”} | Get-ClusteredMailboxServerStatus

9.1.4 Extract Message accept from

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

9.1.5 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

9.1.6 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,MessageSubject, TotalBytes , SourceContext,ConnectorId,Source , InternalMessageId , MessageId ,@{Name=”Recipents”;Expression={$_.recipients}} | export-csv c:\track.csv

9.1.7 Search mailbox / Delete Messages

Searchonly

import-csv c:\tmp\essagesubject.csv | foreach {Search-Mailbox $_.alias -SearchQuery subject:"Test SUbject" -TargetMailbox "Exmon2007" -TargetFolder "Logs" -LogOnly -LogLevel Full} >c:\tmp\output.txt

Delete

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

9.1.8 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,ProhibitSendReceiveQuota,
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
```

9.2. Active Directory

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

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

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

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

http://software.dell.com/trials/

ActiveRoles Management Shell for ActiveDirectory

"A" Product	Trial	Virtual Trial	Freeware	Buy	Contact
Active Administrator	⬇			🛒	✉
Active Administrator for Azure Active Directory	⬇				✉
Active Administrator for Certificate Management	⬇				✉
Active Administrator for DNS Management	⬇				✉
ActiveRoles Management Shell for Active Directory			⬇		
ActiveRoles Server	⬇	⬇			✉

9.2.1 Export Group members - nested / recursive

Just a single line of code will work

```
$members = Get-QADGroupMember  “group Name”  -Indirect |  select Name, Type | Export-Csv .\members.csv
```

9.2.2 Set values for Ad attributes

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

```
Get-QADUser $i.userid | Set-QADUser -ObjectAttributes @{StreetAddress = $i.PhysicalAddress.Trim()}
```

9.2.3 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
$cells=$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:\scripts\exportAD.xlsx"
$workbook.saveas($filepath)
$workbook.close()
$objExcel.Quit()
```

10. Appendix

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])?"

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

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
}
```

Multithreading

```
Start-Job -ScriptBlock {scripttblock } -ArgumentList $arg1,arg2
 While((Get-Job -State 'Running').Count -ge 4)
   {
      Start-Sleep -Milliseconds 10
   }
```