

Basic Tutorials

- <http://computerperformance.co.uk/powershell/>
- [http://www.techotopia.com/index.php/Windows PowerShell 1.0 Essentials](http://www.techotopia.com/index.php/Windows_PowerShell_1.0_Essentials)
- Microsoft's tips and ticks <http://technet.microsoft.com/en-us/library/ee692948.aspx>

Some cool tools

- This is VERY COOL: PowerShell Scriptomatic <http://technet.microsoft.com/en-us/library/ff730935.aspx>

•

Useful Links

- I found a website the purports to have an impersonation script <http://huddledmasses.org/using-alternate-credentials-with-the-filesystem-in-powershell/>
- SQL Server specific scripts <http://www.idera.com/Products/Free-Tools/PowerShell-scripts/>
- PowerShell cmdlet list <http://ss64.com/ps/>
- Lots of scripts <http://powershellcommunity.org/Scripts.aspx>
- SendKeys in PowerShell <http://www.microsoft.com/technet/scriptcenter/topics/winpsh/convert/wshsendkeys.msp>
- Much more complex SendKeys <http://poshcode.org/987>
- Pretty good SendKeys example <http://ss64.com/vb/sendkeys.html>
- SQL Server Links
 - Detailed backup with SQL Server <http://www.mssqltips.com/tip.asp?tip=1862&home>
 - Lots of free SQL Server Scripts <http://www.idera.com/Products/Free-Tools/PowerShell-scripts/>

Useful Commands

- get-acl Get the security descriptor for a resource, such as a file or registry key.
- Get-History Get a list of the previous commands entered during the current session
- Set-Date Change the system time on the computer to a time that you specify.
- get-credential
- start-service -name Fax (also stop-service and restart-service)
- get-Service | where {\$_.Status -eq "Started"}
- get-EventLog system -newest 2000 | where {\$_.entryType -match "Error"}
- get-psDrive - get drives and info about them.
- get-Process – get all processes
- get-service Get-Service | Where-Object {\$_.status -eq "stopped"}
 - Also {\$_.status -eq "started"}
- get-WmiObject get-WmiObject win32_computersystem

- `get-process Get-Process | Where-Object {$_.PrivateMemorySize -gt 50MB} | SELECT ProcessName, PrivateMemorySize, CPU | Format-List`
- Clean up your XML output with `-NoTypeInfoInformation (Get-ChildItem C:\Scripts | ConvertTo-XML -NoTypeInfoInformation) | Get-Member`

Note: to run a script when you are in the same directory you need to `.\scriptname.ps1`

Useful Scripts

- Find OS Version


```
$strCategory = "computer"
$strOS = "Windows*Server*"
$objDomain = New-Object System.DirectoryServices.DirectoryEntry

$objSearcher = New-Object System.DirectoryServices.DirectorySearcher
$objSearcher.SearchRoot = $objDomain

$objSearcher.Filter = ("OperatingSystem=$strOS")

$colPropList = "name"
foreach ($i in $colPropList){$objSearcher.PropertiesToLoad.Add($i)}

$colResults = $objSearcher.FindAll()

foreach ($objResult in $colResults)
{
    $objComputer = $objResult.Properties;
    $objComputer.name
}

```
- **Update Windows**

```
#Only works with PowerShell 2.0

$FileReport = $true
#put your own file here
$FileReportPath = "c:\IT\Windows Update Reports\"
$AutoRestart = $true
$AutoRestartIfPending = $true

$Path = $FileReportPath + "$env:ComputerName" + "_" + (Get-Date -Format dd-MM-yyyy_HH-mm).ToString() + ".html"

#Testing if there are any pending reboots from earlier Windows Update sessions

```

```

if (Test-Path
"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\WindowsUpdate\Auto
Update\RebootRequired"){

#Report to file if enabled
if ($FileReport -eq $true) {
"Invoke-WindowsUpdate was run on $env:ComputerName, and the server $status
`nPlease run Invoke-WindowsUpdate again when the server is rebooted." | Out-File -
FilePath $path
}

#Reboot if autorestart for pending updates is enabled
if ($AutoRestartIfPending) {shutdown.exe /t 0 /r } }
exit

}

#Checking for available updates
$updateSession = new-object -com "Microsoft.Update.Session"
write-progress -Activity "Updating" -Status "Checking available updates"
$updateSession.CreateUpdateSearcher().Search($criteria).Updates
$downloader = $updateSession.CreateUpdateDownloader()
$downloader.Updates = $updates

#If no updates available, do nothing
if ($downloader.Updates.Count -eq "0") {

#Report to file if enabled
if ($FileReport -eq $true) {
"Invoke-WindowsUpdate was run on $env:ComputerName, but no new updates were
found. Please try again later." | Out-File -FilePath $Path
}

}
else
{
#If updates are available, download and install
write-progress -Activity 'Updating' -Status "Downloading $($downloader.Updates.count)
updates"

$criteria="IsInstalled=0 and Type='Software'"
$resultcode= @{"0="Not Started"; 1="In Progress"; 2="Succeeded"; 3="Succeeded With
Errors"; 4="Failed" ; 5="Aborted" }
$result= $downloader.Download()

```

```

if (($Result.Hresult -eq 0) -and (($result.resultCode -eq 2) -or ($result.resultCode -eq 3))) {
    $updatesToInstall = New-object -com "Microsoft.Update.UpdateColl"
    $Updates | where {$_.isdownloaded} | foreach-Object {$updatesToInstall.Add($_) |
out-null
}

```

```

$installer = $updateSession.CreateUpdateInstaller()
$installer.Updates = $updatesToInstall

```

```

write-progress -Activity 'Updating' -Status "Installing $($Installer.Updates.count)
updates"

```

```

$installationResult = $installer.Install()
$Global:counter--1

```

```

$Report = $installer.updates |
                Select-Object -property
Title,EulaAccepted,@{Name='Result';expression={$ResultCode[$installationResult.GetU
pdateResult($Global:Counter++).resultCode ]}},@{Name='Reboot
required';expression={$installationResult.GetUpdateResult($Global:Counter++).RebootR
equired }} | ConvertTo-Html

```

```

#Report to file if enabled
if ($FileReport -eq $true) {
$Report | Out-File -FilePath $path
}

```

```

#Reboot if this is autostart ne necessary
if ($autoRestart -and $installationResult.rebootRequired) { shutdown.exe /t 0 /r }
}
}

```

- **List Installed Software**

```

#replace with your computers name
$strComputer = " ChucksPC "

```

```

$collInstalled = get-wmiobject -class "Win32_Product" -namespace "root\CIMV2" `
-computername $strComputer

```

```

foreach ($objItem in $ collInstalled) {
    write-host "Caption: " $objItem.Caption
    write-host "Description: " $objItem.Description
    write-host "Identifying Number: " $objItem.IdentifyingNumber
    write-host "Installation Date: " $objItem.InstallDate
    write-host "Installation Date 2: " $objItem.InstallDate2
    write-host "Installation Location: " $objItem.InstallLocation
}

```

```

write-host "Installation State: " $objItem.InstallState
write-host "Name: " $objItem.Name
write-host "Package Cache: " $objItem.PackageCache
write-host "SKU Number: " $objItem.SKUNumber
write-host "Vendor: " $objItem.Vendor
write-host "Version: " $objItem.Version
write-host
}

```

- **List Startup Items**

```

#replace with your computers name
$strComputer = " ChucksPC "

```

```

$colStartUp = get-wmiobject -class "Win32_LogicalProgramGroupItem" -namespace
"root\CIMV2" `
-computername $strComputer

```

```

foreach ($objItem in $ colStartUp)
{
write-host "Caption: " $objItem.Caption
write-host "Description: " $objItem.Description
write-host "InstallationDate: " $objItem.InstallDate
write-host "Name: " $objItem.Name
write-host "Status: " $objItem.Status
write-host
}

```

- **Get drives and sizes**

```

$Disk = get-WmiObject win32_logicaldisk
foreach ($Drive in $Disk)
{
$Drive.DeviceID + " - " + [INT] ($Drive.Size / 1048576) + " MB"
}

```

- **Get Time Details**

```

$strComputer = "."

```

```

$colItems = get-wmiobject -class "Win32_LocalTime" -namespace "root\CIMV2" `
-computername $strComputer

```

```

foreach ($objItem in $colItems) {
write-host "Day: " $objItem.Day
write-host "Day Of Week: " $objItem.DayOfWeek
write-host "Hour: " $objItem.Hour
write-host "Milliseconds: " $objItem.Milliseconds
write-host "Minute: " $objItem.Minute
write-host "Month: " $objItem.Month
write-host "Quarter: " $objItem.Quarter
write-host "Second: " $objItem.Second
write-host "Week In Month: " $objItem.WeekInMonth
write-host "Year: " $objItem.Year
}

```

```
write-host  
}
```

- **Create an OU in Active Directory**

```
$StrOUName = Read-Host "Enter New OU Name"  
Clear  
#replace MyDomainName with your domain  
$ObjDomain = [ADSI]"LDAP://dc=MyDomainName,dc=Com"  
$ObjOU = $ObjDomain.Create("OrganizationalUnit", "ou=" + $StrOUName)  
$ObjOU.SetInfo()
```

```
Write-Host $StrOUName + " has been created"
```

- **Create a User**

```
#replace with YOUR DOMAIN INFO  
$ObjOU = [ADSI]"LDAP://dc=MyDomainName,dc=Com"  
$ObjUser = $ObjOU.Create("user", "cn=MyerKen")  
$ObjUser.Put("sAMAccountName", "myerken")  
$ObjUser.SetInfo()
```

- **Create a Group**

```
$ObjOU = [ADSI]"LDAP://dc=MyDomainName,dc=Com"  
$ObjGroup = $ObjOU.Create("group", "cn=Atl-Users")  
$ObjGroup.Put("sAMAccountName", "Atl-Users")  
$ObjGroup.SetInfo()
```

- **Get Processor Info**

```
$computer = "LocalHost"  
$namespace = "root\CIMV2"  
Get-WmiObject -class Win32_Processor -computername $computer -namespace  
$namespace
```

- **Get Bios Info**

```
$computer = "LocalHost"  
$namespace = "root\CIMV2"  
Get-WmiObject -class Win32_BIOS -computername $computer -namespace  
$namespace
```

- **Get users on PC (Remote or local)**

```
#change computer name to match yours  
$strComputer = "ChucksPC"
```

```
$computer = [ADSI](("WinNT://" + $strComputer + ",computer")  
$computer.name
```

```
$Users = $computer.psbases.children | where{$_.psbase.schemaclassname -eq "User"}
```

```
foreach ($member in $Users.psbases.syncroot)  
{ $member.name }
```

- **Create a Share**

```
$FolderPath = "C:\Temp"
```

```
$ShareName = "MyShare"
$Type = 0
$objWMI = [wmiClass] 'Win32_share'
$objWMI.create($FolderPath, $ShareName, $Type)
```

- **Get all computers in a domain**

```
$strFilter = "computer"
```

```
$objDomain = New-Object System.DirectoryServices.DirectoryEntry
```

```
$objSearcher = New-Object System.DirectoryServices.DirectorySearcher
$objSearcher.SearchRoot = $objDomain
$objSearcher.SearchScope = "Subtree"
$objSearcher.PageSize = 1000
```

```
$objSearcher.Filter = "(objectCategory=$strFilter)"
```

```
$colResults = $objSearcher.FindAll()
```

```
foreach ($i in $colResults)
```

```
{
    $objComputer = $i.GetDirectoryEntry()
    Get-WMIObject Win32_BIOS -computername $objComputer.Name
}
```

- **Do an SQL Server Database Backup**

```
System.Reflection.Assembly[]::LoadWithPartialName('Microsoft.SqlServer.SMO') | out-null
```

```
$s = New-Object ('Microsoft.SqlServer.Management.Smo.Server') "LOCALHOST\"
```

```
#Create a Backup object instance with Microsoft.SqlServer.Management.Smo.Backup
$dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
```

```
#Set the Database property to Northwind
```

```
$dbBackup.Database = "Northwind"
```

```
#Add the backup file to the Devices collection and specify File as the backup type
$dbBackup.Devices.AddDevice("D:\PSScripts\backups\TestBack_FULL.bak", "File")
```

```
#Specify the Action property to generate a fullbackup
```

```
$dbBackup.Action="Database"
```

```
#Call the SqlBackup method to generate the backup
```

```
$dbBackup.SqlBackup($s)
```

- **Does a File Exist?**

```
# Setup source and destination files
```

```
$SourceFile = "c:\test\Test.txt";
```

```
$NewFile = "c:\test\Test2.txt";
```

```
# Now - check to see if $Sourcefile exists, and if # it does copy it to $newfile
```

```
if ([System.IO.File]::Exists($SourceFile))
```

```
{
```

```
    [System.IO.File]::Copy($SourceFile, $NewFile)
```

```
    "Source File ($SourceFile) copied to ($NewFile)"
```

```
}
```

```
else
```

```
{
```

```
    "Source file ($Sourcefile) does not exist."
```

```
}
```

- **Copy a file**

```
Copy-Item -Path c:\file1.txt -Destination c:\somefolder\file1.txt
```

- **Excel and PowerShell**

```
objExcel = New-Object -comobject Excel.Application
```

```
$objExcel.Visible = $True
```

```
$objWorkbook = $objExcel.Workbooks.Add()
```

```
$objWorksheet = $objWorkbook.Worksheets.Item(1)
```

```
$objWorksheet.Cells.Item(1,1) = "A value in cell A1."
```

```
$objWorkbook.SaveAs("C:ScriptsTest.xls")
```

```
$objExcel.Quit()
```