50 PowerShell Automation Scripts for IT Professionals

Automate, Optimize, and Secure Microsoft Environments

Swipe Right

By
Ramesh Kumar

@rameshkmenghwar

management and Windows updates to system monitoring and file operations.

By leveraging these automation scripts, you will:

- Save valuable time by eliminating manual, repetitive work.
- Enhance operational efficiency with optimized processes.
- Strengthen your troubleshooting skills across Microsoft environments.
- Gain confidence in customizing scripts for your own IT infrastructure needs.

Whether you’re new to PowerShell or an experienced administrator, this collection will serve as your **go-to toolkit** for everyday IT automation.

Table of Contents

```
# =========================

# 01. GET SYSTEM INFORMATION

# =========================

# Retrieves detailed system information: OS, hardware, and configuration.

Get-ComputerInfo


# =================

# 02. CHECK DISK SPACE

# =================

# Shows filesystem drives and free/used space.

Get-PSDrive -PSProvider FileSystem | Select-Object Name,
@{n='FreeGB';e={[math]::Round($_.Free/1GB,2)}},
@{n='UsedGB';e={[math]::Round(($_.Used/1GB),2)}},
@{n='TotalGB';e={[math]::Round($_.Size/1GB,2)}}


# ===========================

# 03. LIST RUNNING SERVICES

# ===========================

# Lists services that are currently running on the local machine.

Get-Service | Where-Object {$_.Status -eq 'Running'} | Select-Object
Name, DisplayName, Status
```

```
# =================
# 04. RESTART A SERVICE
# =================
# Restart a named service (example: Spooler). Use -WhatIf to test in production.
param($ServiceName = "Spooler")
Restart-Service -Name $ServiceName -Force -ErrorAction Stop
Write-Output "Service '$ServiceName' restarted."


# ==========================
# 05. LIST INSTALLED SOFTWARE
# ==========================
# Exports installed software (use with care; Win32_Product can be slow).
Get-WmiObject -Class Win32_Product | Select-Object Name, Version, Vendor | Sort-Object Name


# =====================
# 06. CHECK WINDOWS UPDATES
# =====================
# Retrieves Windows Update history/log (requires appropriate permissions).
Get-WindowsUpdateLog
```

```
# =======================

# 07. CREATE SCHEDULED TASK

# =======================

# Creates a daily scheduled task to run a PowerShell script at 09:00.

$action = New-ScheduledTaskAction -Execute "PowerShell.exe" -
Argument "-NoProfile -ExecutionPolicy Bypass -File
`\"C:\Scripts\job.ps1`\""

$trigger = New-ScheduledTaskTrigger -Daily -At 9am

Register-ScheduledTask -TaskName "DailyJob" -Action $action -Trigger
$trigger -RunLevel Highest -User "SYSTEM"


# =========================

# 08. CHECK NETWORK CONNECTIONS

# =========================

# Lists active TCP connections and their states.

Get-NetTCPConnection | Select-Object LocalAddress, LocalPort,
RemoteAddress, RemotePort, State, OwningProcess


# ============

# 09. PING TEST

# ============
# Simple network reachability check (4 ICMP packets).
Test-Connection -ComputerName google.com -Count 4 -Quiet


# =====================
# 10. EXPORT EVENT LOGS
# =====================
# Exports newest 200 System events to CSV for analysis.
Get-WinEvent -LogName System -MaxEvents 200 | Select-Object TimeCreated, Id, LevelDisplayName, Message |
  Export-Csv -Path "C:\Reports\SystemEvents.csv" -NoTypeInformation -Encoding UTF8


# =========================
# 11. GET TOP CPU PROCESSES
# =========================
# Shows top 10 processes by CPU time.
Get-Process | Sort-Object CPU -Descending | Select-Object -First 10 Name, Id, CPU, WS


# =========================
# 12. KILL A PROCESS BY NAME
# =========================
# Force stops a process by name (example: notepad). Use with caution.
Stop-Process -Name "notepad" -Force -ErrorAction SilentlyContinue


# =========================
# 13. ENABLE REMOTE DESKTOP
# =========================
# Enables RDP connections on this machine (registry + firewall).
Set-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Control\Terminal Server' -Name "fDenyTSConnections" -Value 0
Enable-NetFirewallRule -DisplayGroup "Remote Desktop"


# =========================
# 14. DISABLE WINDOWS FIREWALL
# =========================
# Disables all firewall profiles (only where safe/approved).
Set-NetFirewallProfile -Profile Domain,Public,Private -Enabled False


# =================================
# 15. START REMOTE POWERSHELL SESSION
# =================================
# Starts an interactive remote session to a host (enter credentials when prompted).
Enter-PSSession -ComputerName "RemoteHostName" -Credential (Get-Credential)


# =================
# 16. ADD A LOCAL USER
# =================
# Creates a local user (Windows 10/Server 2016+). Prompts for secure password.
$securePwd = Read-Host -AsSecureString "Enter password for new user"
New-LocalUser -Name "TestUser" -Password $securePwd -FullName "Test User" -Description "Created by script"


# ====================
# 17. DELETE A LOCAL USER
# ====================
# Removes a local user account if exists.
if (Get-LocalUser -Name "TestUser" -ErrorAction SilentlyContinue) { Remove-LocalUser -Name "TestUser" }


# =========================
# 18. ADD A USER TO GROUP
# ==========================

# Adds a local user to the Administrators group.

Add-LocalGroupMember -Group "Administrators" -Member "TestUser"


# ================================

# 19. EXPORT USER ACCOUNTS TO CSV

# ================================

# Exports local user accounts with basic info.

Get-LocalUser | Select-Object Name, Enabled, LastLogon | Export-Csv -
Path "C:\Reports\LocalUsers.csv" -NoTypeInformation


# ===================================

# 20. RESET USER PASSWORD (ACTIVE DIRECTORY)

# ===================================

# Resets an AD user password (requires AD module and proper
privileges).

Import-Module ActiveDirectory

Set-ADAccountPassword -Identity "john.doe" -Reset -NewPassword
(ConvertTo-SecureString "N3wP@ssw0rd!" -AsPlainText -Force)

Unlock-ADAccount -Identity "john.doe"

Write-Output "Password reset and account unlocked for john.doe"
```

```
# ============================
# 21. LIST DOMAIN USERS (ACTIVE DIRECTORY)
# ============================
# Exports AD users with display name and UPN.
Import-Module ActiveDirectory
Get-ADUser -Filter * -Properties DisplayName, UserPrincipalName |
Select-Object DisplayName, UserPrincipalName |
  Export-Csv -Path "C:\Reports\ADUsers.csv" -NoTypeInformation


# ==========================
# 22. LIST LOCKED-OUT USERS (AD)
# ==========================
# Finds users currently locked out.
Import-Module ActiveDirectory
Search-ADAccount -LockedOut | Select-Object Name, SamAccountName,
LockedOut | Export-Csv "C:\Reports\LockedOutUsers.csv" -
NoTypeInformation


# =====================
# 23. UNLOCK AD USER
# =====================
# Unlocks a specified AD account.
```

```
param([Parameter(Mandatory=$true)][string]$UserSam)

Import-Module ActiveDirectory

Unlock-ADAccount -Identity $UserSam

Write-Output "Unlocked user $UserSam"


# ==========================

# 24. GET GROUP MEMBERSHIP (AD)

# ==========================

# Lists members of a group recursively.

Import-Module ActiveDirectory

Get-ADGroupMember -Identity "Domain Users" -Recursive | Select-
Object Name, SamAccountName, objectClass


# ==========================

# 25. EXPORT AD GROUPS TO CSV

# ==========================

# Exports AD groups and descriptions for auditing/documentation.

Import-Module ActiveDirectory

Get-ADGroup -Filter * -Properties Description | Select-Object Name,
Description |

  Export-Csv -Path "C:\Reports\ADGroups.csv" -NoTypeInformation
```

```
# ==========================
# 26. LIST ALL INSTALLED HOTFIXES
# ==========================
# Retrieves installed hotfixes/KBs for servers (scriptable across servers).
$servers = @("DC1","FILE01")
foreach ($s in $servers) {
  Invoke-Command -ComputerName $s -ScriptBlock { Get-HotFix | Select-Object HotFixID, InstalledOn } |
    Export-Csv -Path "C:\Reports\$s-HotFixes.csv" -NoTypeInformation
}

# ==================================================
# 27. INSTALL WINDOWS UPDATE VIA POWERSHELL (PSWindowsUpdate)
# ==================================================
# Uses PSWindowsUpdate module to scan and install updates. Test in maintenance windows.
Install-Module -Name PSWindowsUpdate -Force -Scope AllUsers
Import-Module PSWindowsUpdate
Get-WindowsUpdate -AcceptAll -Install -AutoReboot

# ==========================
# 28. GET PRINTER INFORMATION
# ==========================

# Lists printers on the local machine or server.

Get-Printer | Select-Object Name, ShareName, PortName, DriverName,
Published


# ==========================

# 29. RESTART PRINT SPOOLER

# ==========================

# Restarts the print spooler service to recover spooler-related issues.

Restart-Service -Name "Spooler" -Force -ErrorAction Stop

Write-Output "Print Spooler restarted."


# ==========================

# 30. MAP NETWORK DRIVE

# ==========================

# Maps a UNC share to a drive letter for the current user persistently.

New-PSDrive -Name "Z" -PSProvider FileSystem -Root "\\fileserver\share"
-Persist -ErrorAction SilentlyContinue

Write-Output "Mapped \\fileserver\share to Z:"


# ==========================

# 31. DISCONNECT NETWORK DRIVE
# ==========================

# Removes a mapped drive.

if (Get-PSDrive -Name "Z" -ErrorAction SilentlyContinue) { Remove-
PSDrive -Name "Z" -Force }


# ==========================

# 32. CHECK OPEN PORTS

# ==========================

# Lists listening TCP ports and associated processes.

Get-NetTCPConnection -State Listen | Select-Object LocalAddress,
LocalPort, OwningProcess |

  ForEach-Object { $_ + @{ProcessName = (Get-Process -Id
$_.OwningProcess -ErrorAction SilentlyContinue).Name } }


# ==========================

# 33. TEST REMOTE PORT

# ==========================

# Tests TCP connectivity to a specific host:port.

param($TestHost = "smtp.office365.com", $TestPort = 587)

Test-NetConnection -ComputerName $TestHost -Port $TestPort -
InformationLevel Detailed
```

```
# =========================
# 34. GET SYSTEM UPTIME
# =========================
# Returns the system last boot time (uptime calculation can be computed from this).
(Get-CimInstance Win32_OperatingSystem).LastBootUpTime


# =========================
# 35. SHUTDOWN / RESTART COMPUTER
# =========================
# Restart the local computer or shut down (use with admin privileges).
# Restart:
Restart-Computer -Force
# Shutdown:
# Stop-Computer -Force


# =========================
# 36. REMOTE COMPUTER REBOOT
# =========================
# Reboots a remote machine and waits until it's available again.
param([string]$Remote = "SERVER01")
```

```
Restart-Computer -ComputerName $Remote -Force -Wait -For PowerShell

Write-Output "$Remote restarted and responsive."


# ====================================

# 37. ENABLE / DISABLE WINDOWS FEATURES

# ====================================

# Enables or disables Windows optional features (example: .NET 3.5).

# Enable:

Enable-WindowsOptionalFeature -Online -FeatureName NetFx3 -All

# Disable example:

# Disable-WindowsOptionalFeature -Online -FeatureName TelnetClient


# ==========================

# 38. GET INSTALLED ROLES (SERVER)

# ==========================

# Lists installed Windows Server roles and features.

Get-WindowsFeature | Where-Object {$_.Installed -eq $true} | Select-
Object DisplayName, Name


# ==========================

# 39. MONITOR CPU USAGE (LIVE)
# ==========================

# Simple live CPU percentage monitor (polling every 2 seconds).

while ($true) {

  $cpu = Get-Counter '\Processor(_Total)\% Processor Time'

  $value = [math]::Round($cpu.CounterSamples[0].CookedValue,2)

  Write-Host "$(Get-Date -Format HH:mm:ss) CPU: $value%"

  Start-Sleep -Seconds 2

}


# ==========================

# 40. MONITOR MEMORY USAGE (LIVE)

# ==========================

# Polls available physical memory and prints used percentage.

while ($true) {

  $os = Get-CimInstance Win32_OperatingSystem

  $freeMB = [math]::Round($os.FreePhysicalMemory/1024,2)

  $totalMB = [math]::Round($os.TotalVisibleMemorySize/1024,2)

  $usedPct = [math]::Round((($totalMB - $freeMB)/$totalMB)*100,2)

  Write-Host "$(Get-Date -Format HH:mm:ss) FreeMB: $freeMB | Used%:
$usedPct"

  Start-Sleep -Seconds 2
}


# ========================

# 41. EXPORT PERFORMANCE LOGS

# ========================

# Captures a performance counter series to CSV for the specified
duration.

$counter = '\Processor(_Total)\% Processor Time'

$samples = 30

Get-Counter -Counter $counter -SampleInterval 1 -MaxSamples $samples
|

  Select-Object -ExpandProperty CounterSamples |

  Select-Object Timestamp, CookedValue |

  Export-Csv -Path "C:\Reports\CPU_Perf_$(Get-Date -Format
yyyyMMdd_HHmm).csv" -NoTypeInformation


# ========================

# 42. BACKUP FILES TO DIRECTORY

# ========================

# Copies source folder contents to a date-stamped backup folder using
robocopy for reliability.

$Source = "C:\Data"
```

```
$DestRoot = "\\backupserver\backups"

$Dest = Join-Path $DestRoot (Get-Date -Format yyyyMMdd)

New-Item -Path $Dest -ItemType Directory -Force | Out-Null

Robocopy $Source $Dest /MIR /Z /R:3 /W:5


# ==========================

# 43. COMPRESS FILES TO ZIP

# ==========================

# Creates a ZIP archive of a folder (uses .NET).

Add-Type -AssemblyName System.IO.Compression.FileSystem

$sourceFolder = "C:\Logs"

$zipFile = "C:\Backups\Logs_$(Get-Date -Format yyyyMMdd).zip"

[System.IO.Compression.ZipFile]::CreateFromDirectory($sourceFolder,
$zipFile)

Write-Output "Created $zipFile"


# ==========================

# 44. EXTRACT ZIP FILE

# ==========================

# Extracts a ZIP archive to a destination folder.

Add-Type -AssemblyName System.IO.Compression.FileSystem
```

```
$zipFile = "C:\Backups\Logs.zip"

$extractTo = "C:\Temp\Logs"

[System.IO.Compression.ZipFile]::ExtractToDirectory($zipFile, $extractTo)

Write-Output "Extracted to $extractTo"


# ==========================

# 45. COPY FILES OVER NETWORK

# ==========================

# Copies files to a remote UNC path with basic retry logic.

$source = "C:\Reports\*"

$dest = "\\fileserver\incoming\reports\"

$maxAttempts = 3; $attempt = 0

while ($attempt -lt $maxAttempts) {

  try {

    Copy-Item -Path $source -Destination $dest -Recurse -Force -
ErrorAction Stop

    Write-Output "Copy succeeded"

    break

  } catch {

    $attempt++; Write-Warning "Attempt $attempt failed: $_"; Start-Sleep -
Seconds 5

  }
}


# ==========================

# 46. SYNC TWO DIRECTORIES

# ==========================

# Mirrors source to destination using Robocopy (fast and resilient).

Robocopy "C:\Source" "D:\Destination" /MIR /Z /R:2 /W:5


# ==========================

# 47. GET FILE HASH (VERIFY INTEGRITY)

# ==========================

# Computes SHA256 hash for a file to verify integrity.

Get-FileHash -Path "C:\Installers\package.exe" -Algorithm SHA256 |
Format-List


# ==========================

# 48. SEARCH FILES BY EXTENSION

# ==========================

# Finds files with a given extension older than N days.

param([string]$Path = "C:\Logs", [string]$Filter = "*.log",
[int]$OlderThanDays = 30)

Get-ChildItem -Path $Path -Recurse -Filter $Filter -File |
  Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-
$OlderThanDays) } |
  Select-Object FullName, Length, LastWriteTime


# ==========================
# 49. DELETE OLD FILES (AUTO-CLEANUP)
# ==========================
# Removes files older than specified days. Use -WhatIf for a dry run.
param([string]$CleanupPath = "C:\Logs", [int]$Days = 30,
[switch]$WhatIf)
Get-ChildItem -Path $CleanupPath -Recurse -File |
  Where-Object { $_.LastWriteTime -lt (Get-Date).AddDays(-$Days) } |
  ForEach-Object {
    if ($WhatIf) { Write-Output "(WhatIf) Would delete $($_.FullName)" }
else { Remove-Item $_.FullName -Force }
  }


# ==========================
# 50. SEND EMAIL ALERT
# ==========================
# Sends a simple SMTP email alert (adjust SMTP server/auth as required).
param(
  [string]$SmtpServer = "smtp.contoso.com",
  [string]$From = "monitor@contoso.com",
  [string]$To = "admin@contoso.com",
  [string]$Subject = "Alert: Condition triggered",
  [string]$Body = "This is an automated alert from PowerShell."
)
Send-MailMessage -SmtpServer $SmtpServer -From $From -To $To -
Subject $Subject -Body $Body -BodyAsHtml
Write-Output "Email sent to $To via $SmtpServer"
```