**Today Assignment – Date: August 13,2025**

**Name: 34733-Prasang Kumar Singh**

**Gmail:** [**thakurprasang23@gmail.com**](mailto:thakurprasang23@gmail.com)

* ***Topics***
* Process check and Delete
* Loop based scipting operations
* Set-Ini Value
* Toolkit Logging location and how its configured
* MSI/MSP Logging
* individual Command logging within Script How to catch return Codes with in PowerShell.
* MSIX Technology Fundamentals with Architecture

**Process check and Delete**

In PowerShell, the concepts of "Process Check" and "Delete" commands are handled through specific cmdlets, and the PowerShell Application Deployment Toolkit (PSADT) is a popular framework for software deployment automation.

**Process Check in PowerShell**

**What is it?**

In PowerShell, process checking is typically done using the Get-Process cmdlet, which lists running processes on the local computer.

**Uses and Purposes**

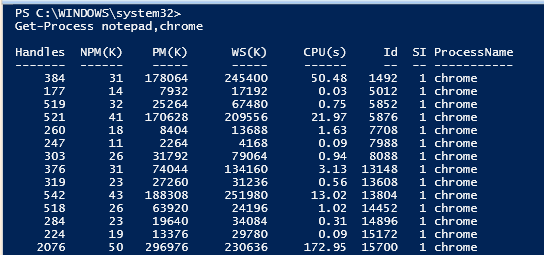
* Monitor running processes.
* Check if a specific process is running.
* Retrieve process details like process ID (PID), CPU usage, memory usage, and more.
* Can be used for scripting tasks where decision-making is based on process availability.

**How It Works**

* Get-Process queries the system for active processes.
* You can filter by process name or other attributes.
* You can terminate processes with cmdlets like Stop-Process.

**Syntax Examples**

* Get all processes: Get-Process
* Get a specific process: Get-Process -Name notepad
* Check if a process exists in a script:



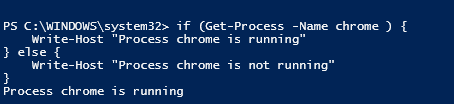
**if** (Get-Process -Name "chrome" -ErrorAction SilentlyContinue) {

Write-Output "Chrome is running"

} **else** {

Write-Output "Chrome is not running"

}



**Delete Command in PowerShell**

**What is it?**

PowerShell uses the Remove-Item cmdlet to delete files, folders, registry keys, and other items supported by various PowerShell providers.

**Uses and Purposes**

* Delete files or directories.
* Remove registry entries.
* Clean up unwanted or obsolete data.
* Automate system maintenance or deployment tasks.

**How It Works**

* Remove-Item deletes the specified item.
* Has parameters for recursive deletion (-Recurse), forced deletion (-Force), and confirmation prompts (-Confirm).
* Supports wildcards to delete multiple items.

**Syntax Examples**

* Delete a file: Remove-Item "C:\path\to\file.txt"
* Delete a folder and its contents: Remove-Item "C:\folder" -Recurse
* Delete with confirmation: Remove-Item "C:\file.txt" -Confirm





**Loop based scipting operations**

The for statement (also known as a for loop) is a language construct you can use to create a loop that runs commands in a command block while a specified condition evaluates to $true.

**Syntax**

The following shows the for statement syntax.

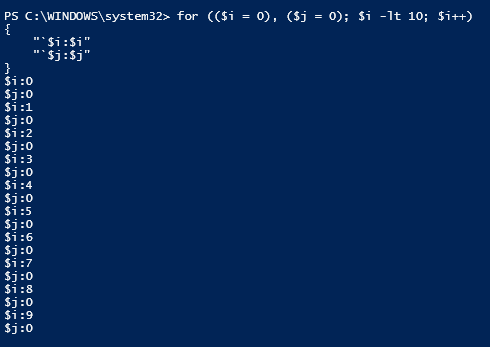
for (<Init>; <Condition>; <Repeat>)

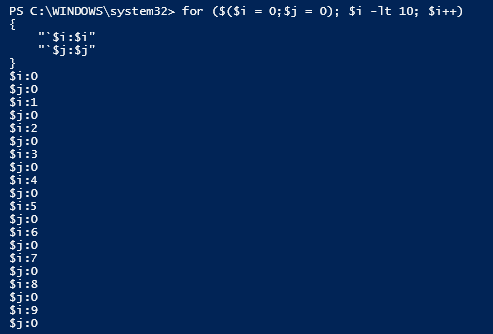
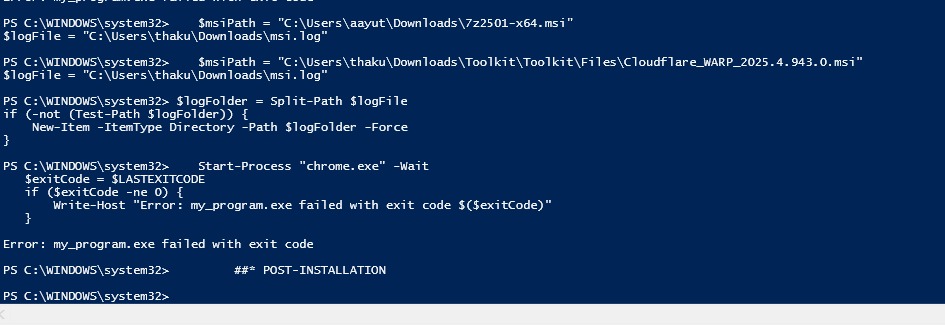
{

<Statement list>

}

**Support for multiple operations**



**Set-IniValue in PowerShell**

**What is it?**

Set-IniValue is a PowerShell function or concept used to update or set a specific key's value in an INI file under a particular section. It may also optionally delete keys or entire sections if desired.

Uses

* Modify existing key-value pairs in INI configuration files.
* Add new keys if they do not exist in the section.
* Delete keys or entire sections from INI files.
* Automate configuration management that uses INI files.

**Syntax Example**

**Set-IniValue -LiteralPath "path\to\file.ini" -Section "SectionName" -Key "KeyName" -Value "NewValue"**

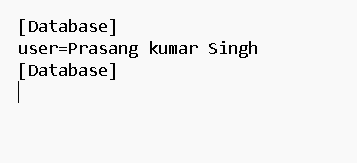
* -LiteralPath: Path to the INI file.
* -Section: The section in the INI file where the key exists or should be added.
* -Key: The key to update or add.
* -Value: The new value for the key.

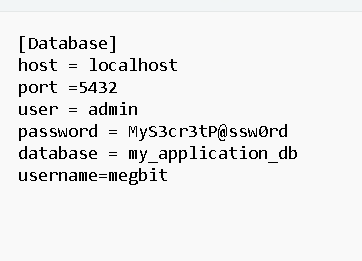
If -Value is omitted, the key can be deleted. If -Key is also omitted, the entire section can be deleted.

**Steps to Use Set-IniValue**

1. Identify the INI file path you want to modify.
2. Specify the section name in the file where the key resides or needs to be created.
3. Specify the key name you want to update or add.
4. Provide the new value to set for that key.
5. Run the Set-IniValue function or script with these parameters.
6. The function reads the INI file, updates or adds the key-value pair under the specified section, and writes changes back to the file.



**Before After**



**Toolkit Logging location and how its configured**

**What is Toolkit Logging?**

Toolkit Logging refers to the methodical capturing of application, system, or toolkit-specific events, errors, and operations into log files by a toolkit or framework. This process is essential for debugging, monitoring, and auditing purposes in software environments. Toolkits such as PSAppDeployToolkit, IBM Toolkits, or integration frameworks often include built-in logging mechanisms to facilitate these functions.

**Logging Location**

* Default Locations:
  + Many toolkits store logs in preset locations by default (e.g., C:\Windows\Logs\Software for PSAppDeployToolkit, or directories like %APPDATA%\Shotgun\logs\ for SG Toolkit).
* **Customizing Location:**
  + You can usually specify an alternate log directory by editing configuration files such as AppDeployToolkitConfig.xml or toolkit-specific config files (e.g., placing <LogFolder>C:\ProgramData\AppInstall-LOGS</LogFolder> in the XML file for PSAppDeployToolkit).
  + Some toolkits allow dynamic path assignment via environment variables or parameters at initialization.

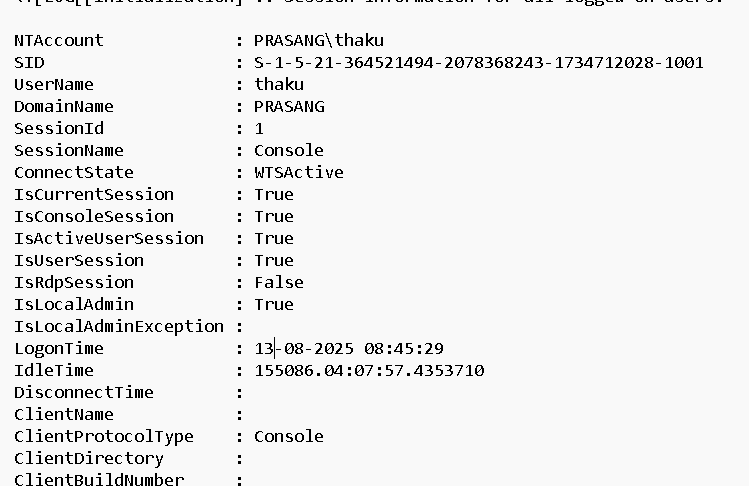
**How is Logging Configured?**

* Configuration Files:
  + Usually managed via XML, JSON, or similar config files (e.g., .xml for PSAppDeployToolkit, .config for .NET, or classpath-included files for adapters).
  + Parameters include the log location, formats, log levels (Trace, Info, Warning, Error, Critical), and message templates.
* Programmatic Configuration:
  + In some languages (e.g., C#/.NET, Java), logging can also be adjusted in code using provided APIs.
  + Advanced toolkits support configuring log rotation, maximum file sizes, log retention, and output to multiple targets (console, file, cloud).
* Logging Levels:
  + Configured to filter the severity of messages recorded (from Debug/Trace to Error and Critical).

**Steps Involved in Toolkit Logging**

Typical steps in a toolkit logging process:

1. Instrument and Collect: Enable logging and define what sources generate log messages.
2. Centralize Storage: Store logs in a consistent directory or cloud platform for later analysis.
3. Configure Parsing: Set up how messages are formatted and parsed (e.g., for easy reading, filtering).
4. Monitor and Alert: Use tools to watch for specific messages/events and generate alerts if certain thresholds are met.
5. Analyze and Report: Use logs for troubleshooting, performance measurement, or compliance reporting.
6. Take Action: Respond to issues revealed by the logs.



**MSI/MSP Logging**

**What is MSI/MSP Logging?**

* MSI Logging involves creating log files during the installation, update, or removal of Windows Installer (.msi) packages. These logs provide a detailed, step-by-step record of all actions the installer takes, including errors and system changes.
* MSP Logging is similar, focusing on the installation of Windows Installer Patch files (.msp), which are used to update or fix installed MSI products. Logging helps troubleshoot patch failures or deployment issues.

**Why Use MSI/MSP Logging?**

* Troubleshooting: Find and diagnose installation issues, errors, or failures.
* Auditing: Keep a record of install, update, or uninstall events for compliance.
* Support: Logs are essential when working with vendor or IT support to resolve deployment problems.
* Transparency: Understand exactly what happened during an install or patch.

Steps Involved in Enabling and Using MSI/MSP Logging

Method : Command-Line Logging

1. Find out the path of your .msi or .msp file (e.g., C:\MyPackage\Example.msi or C:\MyPackage\Patch.msp).
2. Decide the path and filename for your log (e.g., C:\log\example.log).
3. Open Command Prompt on your machine.
4. Run the install or patch using msiexec with logging switches:
   * For MSI Install:

msiexec /i "C:\MyPackage\Example.msi" /L\*V "C:\log\example.log"

* + - /i = Install, /L\*V = Verbose logging, followed by the log file path.
  + For MSP Patch:

text

msiexec /p "C:\MyPackage\Patch.msp" /L\*V "C:\log\patch.log"

* + - /p = Apply patch, /L\*V = Verbose logging.

1. After completion, check the specified log file for details on success or failure



**Individual Command logging within Script**

PSADT provides strong logging features to track every step of a deployment script. The Write-Log function is the main way to log custom messages, command executions, and status updates directly into the toolkit’s main log file.

Logging Methods:

Write-Log – Add custom log entries for messages, commands, or status checks.

Write-Log "Starting installation"

**Execute-Process -Path "msiexec.exe" -Parameters "/i app.msi /qn"**

Write-Log "MSI command executed."

Separate Log for Specific Commands – Some PSADT functions (e.g., Execute-MSI) allow a -LogName parameter to create a dedicated log for that operation.

**Execute-MSI -Action Install -Path "App.msi" -LogName "AppInstall.log"**

PowerShell Built-in Logging – For non-PSADT commands, use Start-Transcript and Stop-Transcript to create separate transcript files.

Benefits:

Clear execution tracking.

Easier troubleshooting.

Proof of installation results.

In short, using Write-Log throughout your PSADT script creates a detailed and easy-to-read record of the deployment process.

**How to catch return Codes with in PowerShell.**

1. Capturing Exit Codes from External Programs: $LASTEXITCODE

* What it does: $LASTEXITCODE is an automatic variable that stores the exit code of the last native (external) command or executable run in PowerShell.
* When to use it: When running .exe, .bat, .cmd, or external binaries — not for native PowerShell cmdlets.

Start-Process "my\_program.exe" -Wait

$exitCode = $LASTEXITCODE

**if** ($exitCode -ne 0) {

Write-Host "Error: my\_program.exe failed with exit code $($exitCode)"

}

* Note: For MSI installs in PSADT, $LASTEXITCODE is especially important since msiexec.exe returns standard Windows Installer error codes (0 = success, others = warnings/errors).

2. Handling Errors in PowerShell Cmdlets: try...catch

* Why needed: Pure PowerShell cmdlets do not set $LASTEXITCODE — they throw exceptions or write errors.
* Use try...catch to trap and handle these errors.

Example:

**try** {

Get-Content "nonexistent\_file.txt"

}

**catch** {

Write-Host "An error occurred: $($\_.Exception.Message)"

}

Important — ErrorActionPreference

$ErrorActionPreference = "Stop" *# Converts non-terminating errors into terminating ones*

**try** {

Get-ChildItem "C:\nonexistent\_folder"

}

**catch** {

Write-Host "Error: $($\_.Exception.Message)"

}

3. Returning Your Own Exit Codes from Scripts

You can set a custom exit code using the exit keyword.

Example:

**if** ($someCondition) {

**exit** 1 *# Failure*

}

**else** {

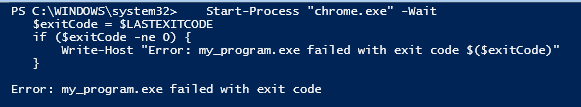
**exit** 0 *# Success*

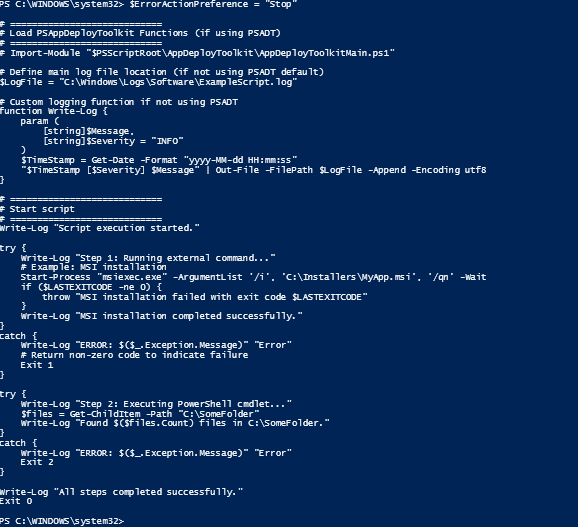
}

You can also pass on the last external command’s code:

my\_program.exe

**exit** $LASTEXITCODE





**MSIX Technology Fundamentals with Architecture**

**MSIX Fundamentals**

1. Modern Packaging – Unified format for both Win32 and UWP apps, replacing MSI/AppX/App-V.
2. Digitally Signed – Every package must be signed to ensure security and integrity.
3. Reliable Install/Uninstall – Clean removal with no leftover files or registry entries.
4. Incremental Updates – Uses block map (AppxBlockMap.xml) for faster, bandwidth-efficient updates.
5. Enterprise-ready – Works with Intune, MECM, Microsoft Store, or direct deployment.

**Architecture Highlights**

1. Containerization – Apps run in a virtualized container that isolates files & registry, reducing conflicts.
2. Core Files –
   * AppxManifest.xml – Package identity/configuration.
   * AppxBlockMap.xml – File integrity & update optimization.
   * AppxSignature.p7x – Digital signature.
3. Resource Virtualization – Registry, file paths, COM, protocol handlers, and shortcuts are managed inside the container.
4. Deployment – OS-managed installation, updating, and clean uninstall.