# CST8202 – Windows Desktop Support

## Lab 4 – PowerShell: Comprehensive Step-by-Step Guide

---

## Table of Contents



---

## Introduction to PowerShell {#introduction}

### What is PowerShell?

PowerShell is a powerful command-line shell and scripting language built on the .NET framework. Unlike traditional command prompts, PowerShell uses **cmdlets** (command-lets) that work with objects rather than just text.

### Key PowerShell Concepts

**Cmdlets:** Small, single-function commands that follow a Verb-Noun naming convention (e.g., `Get-Process`, `Copy-Item`)

**Parameters:** Options that modify how a cmdlet works, always preceded by a dash (e.g., `-Name`, `-Path`)

**Arguments:** The values you provide to parameters (e.g., in `Get-Process -Name notepad`, "notepad" is the argument)

**Pipeline:** The `|` symbol that passes output from one cmdlet to another

**Aliases:** Shortened names for cmdlets (e.g., `ls` is an alias for `Get-ChildItem`)

### Running PowerShell as Administrator

Many operations in this lab require administrative privileges:

1. Click the Start menu
2. Type "PowerShell"
3. Right-click "Windows PowerShell"
4. Select "Run as Administrator"
5. Click "Yes" on the User Account Control prompt

---

## Lab Setup {#setup}

### Prerequisites

- Windows 11 Virtual Machine running in VMware Workstation
- PowerShell launched with Administrator privileges
- Take a VM snapshot before beginning (recommended)

### Creating a Snapshot (Recommended)

1. In VMware, go to VM → Snapshot → Take Snapshot
2. Name it "Before Lab 4"
3. This allows you to revert if something goes wrong

---

## Exercise Solutions {#exercises}

---

### Exercise #1: Finding Aliases for Copy-Item

**Objective:** Learn how to discover existing aliases for cmdlets using Get-Alias

**Background:** PowerShell cmdlets often have multiple aliases to make them easier to use. For example, users familiar with DOS might prefer `copy` while Unix users might prefer `cp`.

**Command Structure:**

```powershell
Get-Alias -Definition Copy-Item
```

**Breaking Down the Command:**

- `Get-Alias` - The cmdlet that retrieves alias information
- `-Definition` - The parameter that specifies we want aliases FOR a specific cmdlet
- `Copy-Item` - The argument (the cmdlet we're looking up)

**Why This Works:**

- This command consists of exactly 1 cmdlet, 1 parameter, and 1 argument as required
- The `-Definition` parameter searches for aliases that point TO the specified cmdlet
- You should see at least 3 results: `copy`, `cp`, and `cpi`

**Expected Output:**

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           copy -> Copy-Item
Alias           cp -> Copy-Item
Alias           cpi -> Copy-Item
```

**Lab Report Entry:**

```
#1: Get-Alias -Definition Copy-Item
```

---

## Exercise #2: Creating a New Alias

**Objective:** Create a custom alias to understand how PowerShell allows personalization

**Background:** While PowerShell has built-in aliases, you can create your own for frequently used commands. This is useful for creating shortcuts that match your workflow.

**Command Structure:**

```powershell
New-Alias -Name Dupe -Value Copy-Item
```

**Breaking Down the Command:**

- `New-Alias` - Cmdlet that creates a new alias
- `-Name Dupe` - Parameter and argument specifying the new alias name
- `-Value Copy-Item` - Parameter and argument specifying what cmdlet the alias points to

**Why This Works:**

- Creates a new shortcut called "Dupe" that executes Copy-Item
- After running this, you can type `Dupe file.txt file_backup.txt` instead of `Copy-Item file.txt file_backup.txt`

**Important Note:** This alias only exists for the current PowerShell session. To make it permanent, you would need to add it to your PowerShell profile.

**Lab Report Entry:**

```
#2: New-Alias -Name Dupe -Value Copy-Item
```

---

### Exercise #3: Verifying the New Alias

**Objective:** Confirm that your custom alias was created successfully

**Command Structure:**

```powershell
Get-Alias -Definition Copy-Item
```

**Why We're Doing This:**

- This is the same command from Exercise #1
- Now you should see 4 aliases instead of 3
- This demonstrates that New-Alias successfully added "Dupe" to the alias list

**Expected Output:**

```
CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           copy -> Copy-Item
Alias           cp -> Copy-Item
Alias           cpi -> Copy-Item
Alias           Dupe -> Copy-Item
```

**Lab Report Entry:**

```
#3: Get-Alias -Definition Copy-Item

[Paste the output showing all 4 aliases here]
```

## Exercise #4: Finding a Newly Added Disk

**Objective:** Learn to identify and work with storage devices using PowerShell

**Background:** Before we can format a disk, we need to identify it. In PowerShell, physical disks are numbered (Disk 0, Disk 1, etc.). New disks typically show as "Offline" status and have no partitions.

**First: Add a 2GB Disk in VMware**

1. Ensure your VM is powered off or use hot-add if enabled
2. In VMware: VM → Settings → Add → Hard Disk
3. Choose "Create a new virtual disk"
4. Specify 2 GB disk size
5. Accept defaults and finish
6. Power on the VM if it was off

**Command Structure:**

```powershell
Get-Disk
```

**Breaking Down the Command:**

- `Get-Disk` - Cmdlet that retrieves information about all physical disks
- No parameters needed for basic listing

**What to Look For:**

- Look for a disk with Size of 2 GB
- The new disk will likely show "Offline" under OperationalStatus
- Note the Number (probably Disk 1 or Disk 2)
- PartitionStyle will show "RAW" (unformatted)

**Expected Output Example:**

```
Number Friendly Name    Serial Number   HealthStatus OperationalStatus Total Size Partition Style
------ -------------    -------------   ------------ ----------------- ---------- ---------------
0      VMware Virtual...                Healthy      Online            60 GB      GPT
1      VMware Virtual...                Healthy      Offline           2 GB       RAW
```

**Alternative Command (More Detailed):**

```powershell
Get-Disk | Where-Object Size -eq 2GB
```

This filters to show only disks that are exactly 2GB.

**Lab Report Entry:**

```
#4: Get-Disk
[Note: The disk number of the new 2GB disk is: X]
```

---

## Exercise #5: Formatting the New Disk

**Objective:** Learn to initialize and format a disk using PowerShell commands

**Background:** Before a disk can be used, it must be initialized, partitioned, and formatted. We'll do this in steps.

**Important:** Replace `X` with your actual disk number from Exercise #4!

### Step 1: Initialize the Disk (if needed)

```powershell
Initialize-Disk -Number X -PartitionStyle GPT
```

### Step 2: Create a Partition

```powershell
New-Partition -DiskNumber X -DriveLetter P -UseMaximumSize
```

### Step 3: Format the Volume (THIS is your Lab Answer for #5)

```powershell
Format-Volume -DriveLetter P -FileSystem NTFS -NewFileSystemLabel PSDisk -Confirm:$false
```

**Breaking Down the Format-Volume Command:**

- `Format-Volume` - The cmdlet that formats a volume
- `-DriveLetter P` - Parameter and argument: assigns drive letter P
- `-FileSystem NTFS` - Parameter and argument: formats as NTFS
- `-NewFileSystemLabel PSDisk` - Parameter and argument: sets volume label
- `-Confirm:$false` - Parameter and argument: skips confirmation prompt

**Count Check:** 1 cmdlet ✓, 4 parameters ✓, 4 arguments ✓

**Why These Choices:**

- **NTFS** - Modern file system with security features, large file support, and journaling
- **Drive Letter P** - As specified; makes it easy to reference
- **Label "PSDisk"** - Identifies this as the PowerShell lab disk
- **Confirm:$false** - Automates the process without prompting

**Lab Report Entry:**

```
#5: Format-Volume -DriveLetter P -FileSystem NTFS -NewFileSystemLabel PSDisk -Confirm:$false
```

---

### Exercise #6: Verifying the Volume Creation

**Objective:** Confirm the volume was created correctly and learn to filter output

**Background:** `Get-Volume` shows all volumes including CD-ROM drives. We need to exclude optical drives to see only our hard disk volumes.

**Command Structure:**

```powershell
Get-Volume | Where-Object DriveType -ne CD-ROM
```

**Breaking Down the Command:**

- `Get-Volume` - Retrieves all volumes
- `|` - Pipeline operator (sends output to next command)
- `Where-Object` - Filters objects based on conditions
- `DriveType -ne CD-ROM` - Condition: DriveType Not Equal to CD-ROM

**Alternative Syntax:**

```powershell
Get-Volume -DriveLetter P
```

This directly gets only the P drive, but the first command is more versatile.

**What to Verify:**

- Drive Letter: P
- FileSystemLabel: PSDisk
- FileSystem: NTFS
- Size: Approximately 2 GB (slightly less due to formatting overhead)
- HealthStatus: Healthy

**Expected Output:**

```
DriveLetter FileSystemLabel FileSystem DriveType HealthStatus SizeRemaining Size
----------- --------------- ---------- --------- ------------ ------------- ----
C           System          NTFS       Fixed     Healthy      50 GB         60 GB
P           PSDisk          NTFS       Fixed     Healthy      1.98 GB       2 GB
```

**Lab Report Entry:**

```
#6: Get-Volume | Where-Object DriveType -ne CD-ROM
```

---

## Exercise #7: Creating a Directory

**Objective:** Learn to create directories using PowerShell with absolute paths

**Background:** PowerShell can create directories just like File Explorer, but with more flexibility and automation potential.

**Finding Your Documents Path:**

Your Documents folder is typically at: `C:\Users\YourUsername\Documents`

To find it automatically:

```powershell
$env:USERPROFILE\Documents
```

**Command Structure:**

```powershell
New-Item -Path "$env:USERPROFILE\Documents\Lab4" -ItemType Directory
```

**Or more explicitly:**

```powershell
New-Item -Path "C:\Users\YourUsername\Documents\Lab4" -ItemType Directory
```

**Breaking Down the Command:**

- **`New-Item`** - Cmdlet that creates new items (files, folders, etc.)
- **`-Path`** - Parameter specifying WHERE to create the item
- **`"$env:USERPROFILE\Documents\Lab4"`** - Argument: the full path to create
  - `$env:USERPROFILE` - Environment variable containing your user folder path
  - `\Documents\Lab4` - Subdirectories to create
- **`-ItemType Directory`** - Parameter and argument: specifies we're creating a folder

**Why Use Environment Variables:**

- `$env:USERPROFILE` adapts to any username automatically
- Makes commands portable across different user accounts
- Avoids hardcoding specific usernames

**Expected Output:**

```
    Directory: C:\Users\YourUsername\Documents

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          9/29/2025   2:30 PM                Lab4
```

The `d----` indicates it's a directory.

**Lab Report Entry:**

```
#7: New-Item -Path "$env:USERPROFILE\Documents\Lab4" -ItemType Directory
```

---

## Exercise #8: Getting PowerShell Version and Redirecting Output

**Objective:** Learn to retrieve system information and redirect output to files

**Background:** Redirection operators save command output to files instead of displaying on screen. This is useful for logging, documentation, and creating reports.

**Part 1: View PowerShell Version**

**Command Structure:**

```powershell
$PSVersionTable
```

**Breaking Down the Command:**

- `$PSVersionTable` - Automatic variable containing PowerShell version information
- This is actually a variable, not a cmdlet, but it displays version info

**Expected Output (3+ lines):**

```
Name                           Value
----                           -----
PSVersion                      5.1.19041.1320
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.19041.1320
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

**Part 2: Redirect to File**

**Command Structure:**

```powershell
$PSVersionTable > "$env:USERPROFILE\Documents\PowerShell.txt"
```

**Breaking Down the Redirection:**

- `>` - Redirection operator (overwrites file if it exists)
- `"$env:USERPROFILE\Documents\PowerShell.txt"` - Destination file path

**Understanding Redirection Operators:**

- `>` - Redirects output, overwrites file
- `>>` - Redirects output, appends to file
- `2>` - Redirects errors only
- `*>` - Redirects all output streams

**Why Redirect:**

- Creates permanent records of command output
- Allows you to process output with other tools
- Essential for automation and logging

**Lab Report Entry:**

```
#8: $PSVersionTable > "$env:USERPROFILE\Documents\PowerShell.txt"
```

---

## Exercise #9: Moving Files with Absolute Paths

**Objective:** Learn to move files using absolute paths and understand file system operations

**Background:** Moving files changes their location. Unlike copying, the original file is removed from the source location.

**Note:** The exercise description says "file created in #9" but means "file created in #8"

**Command Structure:**

```
powershell

Move-Item -Path "C:\Users\YourUsername\Documents\PowerShell.txt" -Destination "C:\Users\YourUsername\Do
```

**Using Environment Variables (Better):**

```
powershell

Move-Item "$env:USERPROFILE\Documents\PowerShell.txt" "$env:USERPROFILE\Documents\Lab4\PowerShell.t
```

**Breaking Down the Command:**

- `Move-Item` - Cmdlet that moves files or directories
- **First argument (Path):** `$env:USERPROFILE\Documents\PowerShell.txt`
  - The SOURCE location (where file currently is)
  - This is an absolute path (starts from root)
- **Second argument (Destination):** `$env:USERPROFILE\Documents\Lab4\PowerShell.txt`
  - The DESTINATION location (where file will move to)
  - Also an absolute path
  - Includes the filename (you can rename during move)

**Absolute vs Relative Paths:**

- **Absolute:** Full path from drive root (C:\Users...)
- **Relative:** Path from current location (.\Lab4\file.txt)

**Understanding Move-Item:**

- The file disappears from Documents folder
- The file appears in Lab4 folder
- File contents and properties remain unchanged
- This is a single operation, not copy-then-delete

**Count Check:** 1 cmdlet ✓, 2 arguments (positional parameters) ✓

**Lab Report Entry:**

```
#9: Move-Item "$env:USERPROFILE\Documents\PowerShell.txt"
"$env:USERPROFILE\Documents\Lab4\PowerShell.txt"
```

---

## Exercise #10: Listing Directory Contents Recursively

**Objective:** Display all files and folders including subfolders

**Background:** Recursive listing shows the complete directory tree, useful for verifying file organization and finding files in nested folders.

**Command Structure:**

```powershell
Get-ChildItem -Path "$env:USERPROFILE\Documents" -Recurse
```

**Or using the alias from Lab 1:**

```powershell
ls "$env:USERPROFILE\Documents" -Recurse
```

**Breaking Down the Command:**

- `Get-ChildItem` - Cmdlet that lists directory contents (like `dir` or `ls`)
- `-Path "$env:USERPROFILE\Documents"` - Specifies which folder to list
- `-Recurse` - Parameter that includes all subfolders

**What You'll See:**

```
Directory: C:\Users\YourUsername\Documents

Mode            LastWriteTime       Length Name
----            -------------       ------ ----
d----       9/29/2025   2:30 PM            Lab4

Directory: C:\Users\YourUsername\Documents\Lab4

Mode            LastWriteTime       Length Name
----            -------------       ------ ----
-a---       9/29/2025   2:45 PM         1234 PowerShell.txt
```

**Understanding the Output:**

- **Mode:** File attributes (d=directory, a=archive, r=readonly, h=hidden, s=system)
- **LastWriteTime:** When file was last modified
- **Length:** File size in bytes
- **Name:** File or folder name

**Lab Report Entry:**

```
#10: Get-ChildItem -Path "$env:USERPROFILE\Documents" -Recurse
```

---

## Exercise #11: Copying with Relative Paths

**Objective:** Learn to use relative paths for file operations

**Background:** Relative paths are based on your current working directory. Understanding them is essential for scripting and efficiency.

**Step 1: Change to Documents Directory**

```powershell
Set-Location "$env:USERPROFILE\Documents"
```

**Verify your location:**

```powershell
Get-Location
```

Should show: `C:\Users\YourUsername\Documents`

**Step 2: Copy File Using Relative Paths**

**Command Structure:**

```powershell
Copy-Item .\Lab4\PowerShell.txt .\PowerShell.txt
```

**Or more explicitly:**

```powershell
Copy-Item -Path .\Lab4\PowerShell.txt -Destination .\
```

**Breaking Down the Command:**

- **`Copy-Item`** - Cmdlet that copies files/folders
- **`.\Lab4\PowerShell.txt`** - First argument (source)
  - `.` means "current directory" (Documents)
  - `\Lab4` means go into Lab4 subfolder
  - `\PowerShell.txt` is the file to copy
- **`.\PowerShell.txt`** - Second argument (destination)
  - `.\` means current directory (Documents)
  - Creates a copy in the parent directory of Lab4

**Understanding Relative Path Symbols:**

- `.` - Current directory
- `..` - Parent directory
- `.\folder` - Subfolder in current directory
- `..\folder` - Subfolder in parent directory

**Visual Representation:**

```
Documents (← You are here)
├── Lab4
│   └── PowerShell.txt (source)
└── PowerShell.txt (destination - copied here)
```

**Why Use Relative Paths:**

- Shorter to type
- More portable (work regardless of username or drive)
- Better for scripts that might run in different environments

**Count Check:** 1 cmdlet ✓, 2 arguments (both relative paths) ✓

**Lab Report Entry:**

```
#11: Copy-Item .\Lab4\PowerShell.txt .\PowerShell.txt
```

---

## Exercise #12: Getting MAC Address with Filtering

**Objective:** Learn to extract specific network information and append to files

**Background:** MAC addresses are unique hardware identifiers for network adapters. Filtering helps display only relevant information.

**Command Structure:**

```powershell
Get-NetAdapter | Select-Object Name, MacAddress >> "$env:USERPROFILE\Documents\Lab4\PowerShell.txt"
```

**Breaking Down the Command:**

- `Get-NetAdapter` - Retrieves network adapter information
- `|` - Pipeline operator
- `Select-Object Name, MacAddress` - Filters to show only Name and MAC columns
- `>>` - Append redirection operator (adds to file without overwriting)
- `"$env:USERPROFILE\Documents\Lab4\PowerShell.txt"` - Destination file

**Understanding Select-Object:**

- Filters which properties (columns) to display
- Without it, you'd see 20+ properties
- Specified properties: `Name`, `MacAddress`

**Expected Output Format (2 columns, 3 lines):**

```
Name           MacAddress
----           ----------
Ethernet       00-0C-29-XX-XX-XX
Wi-Fi          A4-B1-C2-XX-XX-XX
```

**Why Append (`>>`) Instead of Overwrite (`>`):**

- File already contains PowerShell version from Exercise #8
- We want to ADD this information, not replace it
- Appending builds a comprehensive log file

**Alternative Command (Formatting):**

```powershell
Get-NetAdapter | Format-Table Name, MacAddress >> "$env:USERPROFILE\Documents\Lab4\PowerShell.txt"
```

**Lab Report Entry:**

```
#12: Get-NetAdapter | Select-Object Name, MacAddress >>
"$env:USERPROFILE\Documents\Lab4\PowerShell.txt"
```

---

## Exercise #13: Getting IPv4 Address with Filtering

**Objective:** Extract specific IP configuration and understand pipeline filtering

**Background:** Network adapters can have multiple IP addresses (IPv4, IPv6, link-local). We need to filter to show only IPv4 addresses.

**Command Structure:**

```
powershell

Get-NetIPAddress -AddressFamily IPv4 | Where-Object {$_.InterfaceAlias -notlike "*Loopback*"} | Select-Object I
```

**Simpler Alternative:**

```
powershell

Get-NetIPAddress | Where-Object AddressFamily -eq IPv4 | Select-Object InterfaceAlias, IPv4Address >> "$env:US
```

**Breaking Down the Command:**

- **Get-NetIPAddress** - Gets IP address configuration
- **Where-Object AddressFamily -eq IPv4** - Filters to show only IPv4 (not IPv6)
- **Select-Object InterfaceAlias, IPv4Address** - Shows only interface name and IP
- **>>** - Appends to PowerShell.txt in Documents (NOT Lab4)

**Important:** This appends to Documents\PowerShell.txt, creating a SECOND version

**Understanding the Filter:**

- **AddressFamily -eq IPv4** - Equals operator filters to IPv4 only
- **InterfaceAlias** - Friendly name of network adapter
- **IPv4Address** - The actual IP address

**Expected Output (2 columns, 3 lines):**

```
InterfaceAlias      IPv4Address
--------------      -----------
Ethernet          192.168.1.100
Wi-Fi             10.0.0.50
```

**Note the Difference:**

- Exercise #12: Appended to `Lab4\PowerShell.txt`
- Exercise #13: Appends to `Documents\PowerShell.txt`
- This creates TWO different files with different content

**Lab Report Entry:**

```
#13: Get-NetIPAddress | Where-Object AddressFamily -eq IPv4 | Select-Object InterfaceAlias, IPv4Address >> "$env:USERPROFILE\Documents\PowerShell.txt"
```

---

## Exercise #14: Comparing Files

**Objective:** Learn to compare file contents to identify differences

**Background:** The two PowerShell.txt files were created differently. One has MAC addresses appended, the other has IPv4 addresses. Comparing reveals these differences.

**Command Structure:**

```
powershell

Compare-Object (Get-Content "$env:USERPROFILE\Documents\PowerShell.txt") (Get-Content "$env:USERPROFI
```

**Breaking Down the Command:**

- `Compare-Object` - Cmdlet that compares two sets of objects
- `Get-Content "path1"` - First file to compare (wrapped in parentheses)
- `Get-Content "path2"` - Second file to compare (wrapped in parentheses)

**Understanding Get-Content:**

- Reads file contents line by line
- Returns an array of strings (one per line)
- Wrapped in `()` to execute first before comparison

**Expected Output:**

```
InputObject                SideIndicator
-----------                -------------
Name           MacAddress   =>
----           -----------  =>
Ethernet       00-0C-29-... =>
InterfaceAlias     IPv4Address  <=
--------------     -----------  <=
Ethernet       192.168.1.10 <=
```

**Understanding SideIndicator:**

- `=>` - Line exists ONLY in the second file (Lab4\PowerShell.txt)
- `<=` - Line exists ONLY in the first file (Documents\PowerShell.txt)
- `==` - Line exists in BOTH files (use `-IncludeEqual` to show these)

**What This Tells You:**

- First file (Documents) contains IPv4 address information
- Second file (Lab4) contains MAC address information
- Both files share the PSVersion table information (not shown because it's the same)

**Alternative with More Detail:**

```powershell
Compare-Object (Get-Content "$env:USERPROFILE\Documents\PowerShell.txt") (Get-Content "$env:USERPROFI
```

**Lab Report Entry:**

```
#14: Compare-Object (Get-Content "$env:USERPROFILE\Documents\PowerShell.txt") (Get-Content
"$env:USERPROFILE\Documents\Lab4\PowerShell.txt")

[Paste the comparison output here]
```

---

## Exercise #15: Finding Last Boot Time

**Objective:** Learn to extract specific system information using pipelines

**Background:** The last boot time tells you when the system was last restarted. This is useful for troubleshooting and verifying uptime.

**Command Structure:**

```powershell
Get-CimInstance Win32_OperatingSystem | Select-Object LastBootUpTime
```

**Breaking Down the Command:**

- `Get-CimInstance` - Retrieves management information from CIM (Common Information Model)
- `Win32_OperatingSystem` - The WMI class containing OS information
- `|` - Pipeline operator
- `Select-Object LastBootUpTime` - Extracts only the boot time property

**Understanding WMI/CIM:**

- **WMI (Windows Management Instrumentation):** Windows system for managing and monitoring
- **CIM (Common Information Model):** Standard for managing hardware and software
- `Win32_OperatingSystem` class contains hundreds of properties about your OS
- We're extracting just one: `LastBootUpTime`

**Alternative Commands:**

**Option 1: More readable format**

```powershell
(Get-CimInstance Win32_OperatingSystem).LastBootUpTime
```

**Option 2: Calculate uptime**

```powershell
$bootTime = (Get-CimInstance Win32_OperatingSystem).LastBootUpTime
$uptime = (Get-Date) - $bootTime
Write-Host "System booted at: $bootTime"
Write-Host "Uptime: $($uptime.Days) days, $($uptime.Hours) hours, $($uptime.Minutes) minutes"
```

**Option 3: Using Get-Uptime (PowerShell 6+)**

```powershell
Get-Uptime -Since
```

**Expected Output:**

```
LastBootUpTime
--------------
9/29/2025 8:15:32 AM
```

**Why This Matters:**

- Verifies if system has been restarted recently
- Useful for troubleshooting (many issues require restart)
- Important for maintenance windows and update verification
- Can indicate system stability (frequent reboots may signal problems)

**Lab Report Entry:**

```
#15: Get-CimInstance Win32_OperatingSystem | Select-Object LastBootUpTime
```

---

## Bonus Challenge: Creating a Mirrored Storage Pool {#bonus}

**Objective:** Learn advanced storage management by creating RAID-like mirrored storage

**Background:** Storage Spaces is a Windows feature that pools multiple physical disks and creates virtual disks with resiliency. A mirror provides redundancy—if one disk fails, data remains accessible on the other.

**Important:** Take a VM snapshot BEFORE starting this section!

---

### Step 1: Add Two 5GB Disks in VMware

1. Power off your VM
2. VM → Settings → Add → Hard Disk
3. Create new virtual disk, 5 GB
4. Repeat to add a second 5 GB disk
5. Power on VM
6. Launch PowerShell as Administrator

---

### Step 2: Identify the New Disks

```powershell
Get-PhysicalDisk | Where-Object CanPool -eq $true
```

**What This Does:**

- Lists all physical disks that can be added to a storage pool
- New, unformatted disks will show `CanPool = True`
- Note the `UniqueId` or `FriendlyName` of both disks

**Expected Output:**

```
FriendlyName   CanPool  OperationalStatus  HealthStatus  Usage     Size
------------   -------  -----------------  ------------  -----     ----
PhysicalDisk1  True     OK                 Healthy       Auto-Select 5 GB
PhysicalDisk2  True     OK                 Healthy       Auto-Select 5 GB
```

---

## Step 3: Create the Storage Pool

### Complete Command:

```powershell
New-StoragePool -FriendlyName "MirrorPool" -StorageSubsystemFriendlyName "Windows Storage*" -PhysicalD
```

### Breaking Down the Command:

- **`New-StoragePool`** - Creates a new storage pool
- **`-FriendlyName "MirrorPool"`** - Names your pool
- **`-StorageSubsystemFriendlyName "Windows Storage*"`** - Specifies the storage subsystem
- **`-PhysicalDisks (Get-PhysicalDisk -CanPool $true)`** - Automatically includes all poolable disks

### What Happens:

- Both 5GB disks are combined into a single storage pool
- The pool can now be used to create virtual disks
- Raw capacity: 10GB total

---

## Step 4: Create a Mirrored Virtual Disk

```powershell
New-VirtualDisk -StoragePoolFriendlyName "MirrorPool" -FriendlyName "MirroredDisk" -ResiliencySettingName
```

### Breaking Down the Command:

- **`New-VirtualDisk`** - Creates a virtual disk from the storage pool
- **`-StoragePoolFriendlyName "MirrorPool"`** - Uses the pool we just created
- **`-FriendlyName "MirroredDisk"`** - Names the virtual disk
- **`-ResiliencySettingName "Mirror"`** - Specifies mirroring (RAID 1)
- **`-Size 5GB`** - Creates a 5GB virtual disk (uses 10GB raw capacity due to mirroring)

**Understanding Mirroring:**

- Data is written to both disks simultaneously
- If one disk fails, data remains accessible on the other
- Usable space is half of total raw capacity (5GB usable from 10GB raw)
- Similar to RAID 1 in traditional storage

**Expected Output:**

```
FriendlyName ResiliencySettingName OperationalStatus HealthStatus Size
------------ --------------------- ----------------- ------------ ----
MirroredDisk Mirror                OK                Healthy      5 GB
```

---

## Step 5: Initialize and Format the Virtual Disk

**Initialize the disk:**

```powershell
Get-VirtualDisk -FriendlyName "MirroredDisk" | Get-Disk | Initialize-Disk -PartitionStyle GPT
```

**Create partition and format:**

```powershell
Get-VirtualDisk -FriendlyName "MirroredDisk" | Get-Disk | New-Partition -DriveLetter M -UseMaximumSize | Form
```

**What This Pipeline Does:**

1. **Get-VirtualDisk** - Retrieves the virtual disk we created
2. **Get-Disk** - Gets the associated physical disk object
3. **New-Partition** - Creates a partition using all available space with drive letter M
4. **Format-Volume** - Formats it as NTFS with the label "MirroredVolume"

---

## Step 6: Verify the Complete Setup

**Check the storage pool:**

```powershell
Get-StoragePool -FriendlyName "MirrorPool" | Format-List
```

**Check the virtual disk:**

```powershell
Get-VirtualDisk -FriendlyName "MirroredDisk" | Format-List
```

**Check the volume:**

```powershell
Get-Volume -DriveLetter M
```

**Expected Results:**

- Storage pool should show 10GB total capacity
- Virtual disk should show Mirror resiliency and Healthy status
- Volume M: should be accessible with ~5GB capacity

---

## Complete Bonus Command Pipeline

**The single-line solution for the bonus:**

```powershell
New-StoragePool -FriendlyName "MirrorPool" -StorageSubsystemFriendlyName "Windows Storage*" -PhysicalD
```

**Breaking Down This Pipeline:**

1. Creates storage pool from all available disks
2. Pipes pool to create mirrored virtual disk
3. Gets the disk object
4. Initializes with GPT partition style
5. Creates partition with drive letter M
6. Formats as NTFS

**Lab Report Entry:**

```
Bonus: New-StoragePool -FriendlyName "MirrorPool" -StorageSubsystemFriendlyName "Windows Storage*" -PhysicalDisks (Get-PhysicalDisk -CanPool $true) | New-VirtualDisk -FriendlyName "MirroredDisk" -ResiliencySettingName "Mirror" -Size 5GB | Get-Disk | Initialize-Disk -PartitionStyle GPT -PassThru | New-Partition -DriveLetter M -UseMaximumSize | Format-Volume -FileSystem NTFS -NewFileSystemLabel "MirroredVolume" -Confirm:$false
```

## Submission Guidelines {#submission}

### Lab Report Format

Your lab report should include the following for each exercise:

**Format for Written Answers (#):**

- Exercise number
- The complete PowerShell command you used
- Brief explanation if required

**Example:**

```
#1: Get-Alias -Definition Copy-Item
```

**Format for Screenshots (%):**

- Clear, legible screenshot showing the command and its output
- Include the PowerShell prompt to show context
- Highlight or annotate if necessary

---

### Checklist Before Submission

✅ **All exercises completed (#1-#15)**

- [ ] Exercise #1: Get-Alias command
- [ ] Exercise #2: New-Alias command
- [ ] Exercise #3: Output showing 4 aliases
- [ ] Exercise #4: Get-Disk command and disk number identified
- [ ] Exercise #5: Format-Volume command with 4 parameters
- [ ] Exercise #6: Get-Volume command excluding CD-ROM
- [ ] Exercise #7: New-Item command for Lab4 directory
- [ ] Exercise #8: $PSVersionTable redirection command
- [ ] Exercise #9: Move-Item command with absolute paths
- [ ] Exercise #10: Get-ChildItem recursive command
- [ ] Exercise #11: Copy-Item command with relative paths
- [ ] Exercise #12: Get-NetAdapter pipeline with MAC addresses
- [ ] Exercise #13: Get-NetIPAddress pipeline with IPv4
- [ ] Exercise #14: Compare-Object output
- [ ] Exercise #15: Get-CimInstance boot time command

✅ **Bonus (if attempted)**

- [ ] Complete storage pool creation pipeline

✅ **Formatting Requirements**

- [ ] Commands are clearly labeled with exercise numbers
- [ ] Output/screenshots are provided where requested
- [ ] Document follows lab instructions formatting
- [ ] All commands are tested and verified working

---

## Additional Tips and Best Practices

### PowerShell Best Practices

1. **Use Tab Completion:** Press Tab after typing part of a cmdlet or parameter name to auto-complete
2. **Get Help:** Use `Get-Help <cmdlet-name>` to learn about any cmdlet
3. **Check Command History:** Use up/down arrows to cycle through previous commands
4. **Use `-WhatIf`:** Add this parameter to see what a command would do without actually doing it
5. **Experiment Safely:** Always take VM snapshots before major changes

### Common Errors and Solutions

**Error: "Cannot bind parameter" or "Parameter set cannot be resolved"**

- **Solution:** Check your parameter spelling and ensure you're using compatible parameters together

**Error: "Access Denied" or "Insufficient Permissions"**

- **Solution:** Ensure PowerShell is running as Administrator

**Error: "File not found" or "Path does not exist"**

- **Solution:** Verify your path syntax and use `Get-Location` to check current directory

**Error: "Disk is offline"**

- **Solution:** Use `Set-Disk -Number X -IsOffline $false` to bring disk online

**Error: "The requested operation requires elevation"**

- **Solution:** Right-click PowerShell and select "Run as Administrator"

---

## Useful PowerShell Cmdlets Reference

**File System Operations:**

- `Get-ChildItem` (alias: `ls`, `dir`) - List directory contents
- `Set-Location` (alias: `cd`) - Change directory
- `Copy-Item` (alias: `copy`, `cp`) - Copy files/folders
- `Move-Item` (alias: `move`, `mv`) - Move files/folders
- `Remove-Item` (alias: `del`, `rm`) - Delete files/folders
- `New-Item` - Create new files/folders

**Storage Management:**

- `Get-Disk` - List all disks
- `Get-Partition` - List all partitions
- `Get-Volume` - List all volumes
- `Initialize-Disk` - Prepare a disk for use
- `New-Partition` - Create a partition
- `Format-Volume` - Format a volume

**Network Operations:**

- `Get-NetAdapter` - Show network adapters
- `Get-NetIPAddress` - Show IP addresses
- `Test-Connection` (similar to `ping`) - Test network connectivity

**System Information:**

- `Get-ComputerInfo` - Comprehensive system information
- `Get-CimInstance` - Query WMI/CIM classes
- `Get-Process` - List running processes
- `Get-Service` - List services

**Text Processing:**

- `Select-Object` - Select specific properties
- `Where-Object` - Filter objects
- `Sort-Object` - Sort objects
- `Format-Table` - Format output as table
- `Format-List` - Format output as list

---

## Learning Resources

### Official Microsoft Documentation

- PowerShell Documentation
- PowerShell Cmdlet Reference
- Storage Spaces Documentation

### PowerShell Learning Path

1. Learn basic cmdlets and pipeline
2. Understand objects and properties
3. Master filtering and selection
4. Practice with file system operations
5. Explore system administration tasks
6. Study scripting and automation

### Quick Reference Cards

- **Get help on any cmdlet:** `Get-Help <cmdlet> -Examples`
- **Find cmdlets:** `Get-Command *keyword*`
- **View cmdlet parameters:** `Get-Command <cmdlet> -Syntax`
- **View object properties:** `Get-<cmdlet> | Get-Member`

---

## Troubleshooting Guide

### VMware Issues

#### Problem: Cannot add new disk

- Ensure VM is powered off (unless hot-add is enabled)
- Check available disk space on host machine
- Verify VMware Workstation has necessary permissions

#### Problem: Disk not showing in PowerShell

- Refresh disk management: `Update-HostStorageCache`
- Rescan for hardware changes
- Restart the VM

### PowerShell Issues

#### Problem: Commands not recognized

- Verify correct PowerShell version: `$PSVersionTable`
- Check module availability: `Get-Module -ListAvailable`
- Import required module: `Import-Module <ModuleName>`

#### Problem: Execution policy errors

- Check policy: `Get-ExecutionPolicy`
- Set policy (as Admin): `Set-ExecutionPolicy RemoteSigned`

#### Problem: Pipeline errors

- Test each part of pipeline separately
- Use `Get-Member` to see available properties
- Verify object types match between pipeline stages

---

## Glossary

**Alias:** A shorthand name for a cmdlet (e.g., `ls` for `Get-ChildItem`)

**Argument:** A value provided to a parameter (e.g., in `-Name "test"`, "test" is the argument)

**CIM (Common Information Model):** A standard for managing hardware and software information

**Cmdlet:** A PowerShell command following Verb-Noun naming (e.g., `Get-Process`)

**GPT (GUID Partition Table):** Modern partitioning scheme supporting large disks

**MBR (Master Boot Record):** Legacy partitioning scheme with limitations

**NTFS (New Technology File System):** Windows file system with security and large file support

**Parameter:** An option that modifies cmdlet behavior (e.g., `-Name`, `-Path`)

**Pipeline:** Connecting cmdlets with `|` to pass output as input

**RAW:** Unformatted disk with no partition table

**Redirection:** Using `>` or `>>` to send output to files

**Resiliency:** Fault tolerance in Storage Spaces (Mirror, Parity, Simple)

**Storage Pool:** Collection of physical disks managed as single unit

**Virtual Disk:** Logical disk created from storage pool

**WMI (Windows Management Instrumentation):** Windows infrastructure for managing systems

---

## Conclusion

This lab has introduced you to fundamental PowerShell concepts including:

- Discovering and creating aliases
- Managing disks, partitions, and volumes
- Working with files and directories using both absolute and relative paths
- Filtering and selecting specific information from cmdlet output
- Redirecting and appending output to files
- Comparing file contents
- Extracting system and network information
- (Bonus) Creating resilient storage with Storage Spaces

PowerShell is an essential skill for Windows system administrators. Practice these commands and experiment with variations to deepen your understanding. As you become more comfortable, explore scripting to automate repetitive tasks.

**Remember:** Always test commands in a safe environment (like your VM) before using them in production, and maintain good snapshot hygiene for easy recovery.

---

**Document Version:** 1.0
**Last Updated:** September 29, 2025
**Course:** CST8202 – Windows Desktop Support
**Lab:** Lab 4 – PowerShell