# Lab Guide 1: Installing PowerShell

#### Objective:

Learn how to install PowerShell on various platforms.

#### Instructions:

\*\*Windows:\*\*

1. Open a web browser and navigate to the [PowerShell GitHub releases page](https://github.com/PowerShell/PowerShell/releases).

2. Download the latest `.msi` package for Windows.

3. Run the installer and follow the prompts:

- Accept the license agreement.

- Choose the installation directory.

- Select additional options as needed (e.g., adding PowerShell to the PATH environment variable).

4. Once the installation is complete, open PowerShell by searching for "PowerShell" in the Start menu and selecting "Windows PowerShell" or "PowerShell".

\*\*Linux:\*\*

1. Open a terminal.

2. Depending on your Linux distribution, follow the appropriate commands to install PowerShell:

\*\*Debian/Ubuntu:\*\*

```bash

sudo apt update

sudo apt install -y wget apt-transport-https software-properties-common

wget -q https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

sudo apt update

sudo apt install -y powershell

```

Start PowerShell by typing `pwsh`.

\*\*CentOS/RHEL:\*\*

```bash

sudo yum install -y wget

sudo wget -q https://packages.microsoft.com/config/rhel/7/packages-microsoft-prod.rpm

sudo rpm -Uvh packages-microsoft-prod.rpm

sudo yum install -y powershell

```

Start PowerShell by typing `pwsh`.

\*\*macOS:\*\*

1. Open a terminal.

2. Install Homebrew if it is not already installed:

```bash

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

```

3. Use Homebrew to install PowerShell:

```bash

brew update

brew install --cask powershell

```

Start PowerShell by typing `pwsh`.

#### Verification:

1. Open PowerShell by typing `powershell` (Windows) or `pwsh` (Linux/macOS).

2. Verify the installation by checking the version:

```powershell

$PSVersionTable.PSVersion

```

# Lab Guide 2: Exploring Cmdlets

#### Objective:

Learn how to use Get-Help, Get-Command, and Get-Member cmdlets.

#### Instructions:

1. \*\*Using Get-Help:\*\*

- Open PowerShell.

- Display help for a cmdlet:

```powershell

Get-Help Get-Process

```

- Display detailed help:

```powershell

Get-Help Get-Process -Detailed

```

- Update the help content:

```powershell

Update-Help

```

2. \*\*Using Get-Command:\*\*

- List all available cmdlets:

```powershell

Get-Command

```

- Find a specific cmdlet:

```powershell

Get-Command \*service\*

```

3. \*\*Using Get-Member:\*\*

- Retrieve properties and methods of an object:

```powershell

Get-Process | Get-Member

```

#### Exercises:

1. Use `Get-Help` to find information about the `Get-Service` cmdlet.

2. Use `Get-Command` to find cmdlets related to files.

3. Use `Get-Member` to explore properties and methods of the `Get-Service` output.

# Lab Guide 3: Using the Pipeline

#### Objective:

Learn how to filter and sort data using the PowerShell pipeline.

#### Instructions:

1. \*\*Filtering Data:\*\*

- Open PowerShell.

- Use `Where-Object` to filter processes:

```powershell

Get-Process | Where-Object { $\_.CPU -gt 10 }

```

2. \*\*Sorting Data:\*\*

- Sort processes by CPU usage:

```powershell

Get-Process | Sort-Object -Property CPU -Descending

```

3. \*\*Selecting Data:\*\*

- Select specific properties of processes:

```powershell

Get-Process | Select-Object -Property Name, CPU

```

#### Exercises:

1. List all services that are currently running.

2. Sort the list of running processes by memory usage.

3. Select the name and status of all services and display them.

# Lab Guide 4: Variables and Arrays

#### Objective:

Learn how to create and manipulate variables, arrays, and hash tables.

#### Instructions:

1. \*\*Creating Variables:\*\*

- Open PowerShell.

- Create a variable:

```powershell

$greeting = "Hello, PowerShell!"

```

2. \*\*Creating Arrays:\*\*

- Create an array:

```powershell

$numbers = 1, 2, 3, 4, 5

```

3. \*\*Creating Hash Tables:\*\*

- Create a hash table:

```powershell

$person = @{

Name = "John Doe"

Age = 30

Occupation = "Developer"

}

```

4. \*\*Manipulating Variables:\*\*

- Access array elements:

```powershell

$numbers[0]

```

- Add elements to an array:

```powershell

$numbers += 6

```

- Access hash table values:

```powershell

$person["Name"]

```

#### Exercises:

1. Create a variable to store your name.

2. Create an array of the first five prime numbers.

3. Create a hash table to store information about your favorite book.

# Lab Guide 5: Writing Scripts

#### Objective:

Learn how to write and run basic PowerShell scripts.

#### Instructions:

1. \*\*Writing a Script:\*\*

- Open a text editor (e.g., Notepad).

- Write a simple script:

```powershell

# My first script

$greeting = "Hello, PowerShell!"

Write-Output $greeting

```

2. \*\*Saving a Script:\*\*

- Save the file with a `.ps1` extension, e.g., `MyScript.ps1`.

3. \*\*Running a Script:\*\*

- Open PowerShell.

- Navigate to the directory where the script is saved.

- Run the script:

```powershell

.\MyScript.ps1

```

4. \*\*Handling Errors:\*\*

- Add error handling to your script:

```powershell

try {

Get-Process -Name "NonExistentProcess"

} catch {

Write-Output "Process not found."

}

```

#### Exercises:

1. Write a script that prints "Hello, [Your Name]!".

2. Modify the script to handle cases where a non-existent process is queried.

3. Save and run your script.

# Lab Guide 6: Remoting and Automation

#### Objective:

Learn how to set up PowerShell remoting and automate tasks.

#### Instructions:

1. \*\*Setting Up Remoting:\*\*

- Open PowerShell with administrative privileges.

- Enable remoting:

```powershell

Enable-PSRemoting -Force

```

2. \*\*Using Remoting:\*\*

- Start a remote session:

```powershell

Enter-PSSession -ComputerName RemoteComputerName

```

- Run a command on a remote computer:

```powershell

Invoke-Command -ComputerName RemoteComputerName -ScriptBlock { Get-Process }

```

3. \*\*Automating Tasks:\*\*

- Create a scheduled task:

```powershell

$action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument '-File C:\Scripts\MyScript.ps1'

$trigger = New-ScheduledTaskTrigger -Daily -At 7am

Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "MyDailyTask"

```

#### Exercises:

1. Enable remoting on your machine.

2. Start a remote session to another computer (use a virtual machine if needed).

3. Create a scheduled task to run a script at a specific time.

# Lab Guide 7: Working with Modules

#### Objective:

Learn how to install and create PowerShell modules.

#### Instructions:

1. \*\*Installing Modules:\*\*

- Open PowerShell.

- Find a module in the PowerShell Gallery:

```powershell

Find-Module -Name PSReadLine

```

- Install the module:

```powershell

Install-Module -Name PSReadLine

```

2. \*\*Using Modules:\*\*

- Import a module:

```powershell

Import-Module -Name PSReadLine

```

3. \*\*Creating Modules:\*\*

- Create a module file:

```powershell

function Get-Greeting {

param ($Name)

"Hello, $Name!"

}

Export-ModuleMember -Function Get-Greeting

```

- Save the file with a `.psm1` extension, e.g., `MyModule.psm1`.

4. \*\*Using Custom Modules:\*\*

- Import your custom module:

```powershell

Import-Module -Name C:\Path\To\MyModule.psm1

```

- Use the function from your module:

```powershell

Get-Greeting -Name "PowerShell"

```

#### Exercises:

1. Install a module from the PowerShell Gallery.

2. Create a custom module with a function that prints a greeting.

3. Import and use your custom module.