ADMINISTRATION OF OPERATING SYSTEMS

PowerShell

Windows PowerShell Introduction

- PowerShell is an automation solution that consists of:
  - A command-line shell.
  - A scripting language.
  - A configuration management framework.
- Commands include:
  - Cmdlets
  - Functions
  - Filters
  - Workflows
- PowerShell was originally built on the .NET framework and only worked on Windows operating systems.
- It currently uses the .NET Core and can run on Windows, MacOS, and Linux platforms.

Windows PowerShell versions

Version	Release date	Notes
PowerShell 7.2	November 2021	Built on .NET 6.0.
PowerShell 7.1	November 2020	Built on .NET 5.0.
PowerShell 7.0	March 2020	Built on .NET Core 3.1.
PowerShell 6.0	September 2018	Built on .NET Core 2.0. First release that's installable on Windows, Linux, and macOS.
PowerShell 5.1	August 2016	Released in Windows 10 Anniversary Update and Windows Server 2016 and as part of Windows Management Framework (WMF) 5.1.
PowerShell 5.0	February 2016	Integrated in Windows 10 version 1511. Released in Windows Management Framework (WMF) 5.0. Can be installed on Windows Server 2008 R2, Windows Server 2012, Windows 10, Windows 8.1 Enterprise, Windows 8.1 Pro, and Windows 7 SP1.
PowerShell 4.0	October 2013	Integrated in Windows 8.1 and Windows Server 2012 R2. Can be installed on Windows 7 SP1, Windows Server 2008 SP1, and Windows Server 2012.

Windows PowerShell versions (Slide 2)

Version	Release date	Notes
PowerShell 3.0	October 2012	Integrated in Windows 8 and Windows Server 2012. Can be installed on Windows 7 SP1, Windows Server 2008 SP1, and Windows Server 2008 R2 SP1.
PowerShell 2.0	July 2009	Integrated in Windows 7 and Windows Server 2008 R2. Can be installed on Windows XP SP3, Windows Server 2003 SP2, and Windows Vista SP1.
PowerShell 1.0	November 2006	Installable on Windows XP SP2, Windows Server 2003 SP1, and Windows Vista. Optional component of Windows Server 2008.

Windows PowerShell applications

- Windows PowerShell console includes:
  - Basic command-line interface.
  - Maximum support for PowerShell features.
  - Minimal editing capabilities.
- Windows PowerShell ISE includes:
  - Script editor and console combination.
  - Rich editing capabilities.
- PowerShell Core doesn’t support Windows PowerShell ISE. It uses VS Code with the PowerShell extension.

Considerations when using PowerShell

- When using PowerShell, you should:
  - Install and use PowerShell side-by-side with Windows PowerShell:
    - PowerShell uses a separate installation path and executable name (pwsh.exe).
    - PowerShell uses a separate PSModulePath, profile, and event logs.
    - You identify the PowerShell version by using **$PSVersionTable**.
  - Run PowerShell using Administrative credentials:
    - 64-bit operating systems include both 64-bit and 32-bit versions of PowerShell.
    - The Windows title bar must display **Administrator** if you need administrative privileges in Windows PowerShell.
    - When UAC is enabled, you must right-click the application icon or activate its context menu to run as Administrator.
  - Identify and modify the execution policy in PowerShell:
    - Use **Get-ExecutionPolicy** to identify the effective execution policy in PowerShell.
    - Be aware that **Restricted** is the default for Windows clients and **RemoteSigned** is the default for Windows servers.

Configuring the PowerShell console

- Select a font style, size, and color, and set the screen color so that text is easy to read and you can differentiate between often-confused characters, such as ` ' ({ [<.
- Modify screen size to maximize available space for output.
- Make sure that the screen buffer width is smaller than window width.
- Enable copy and paste.

Demonstration: Configuring the console

- In this demonstration, you will learn how to:

1. Run the 64-bit console as Administrator.
2. Set a font family.
3. Set a console layout.
4. Start a transcript.

Configuring the ISE

- Two panes: script and console.
- One-pane and two-pane view options.
- **Command Add-on** displays available commands.
- Customization of font style, size, and color.
- Customization of screen color.
- Bundling of color selections into themes.
- Additional features include snippets, add-ins, and debugging.

Cmdlet	Description
Get-ComputerInfo	Retrieves all system and operating system properties from the computer
Get-Service	Retrieves a list of all services on the computer
Get-EventLog	Retrieves events and event logs from local and remote computers (Only available in Windows PowerShell 5.1)
Get-Process	Retrieves a list of all active processes on a local or remote computer
Stop-Service	Stops one or more running services
Stop-Process	Stops one or more running processes
Stop-Computer	Shuts down local and remote computers
Clear-EventLog	Deletes all of the entries from the specified event logs on the local computer or on remote computers
Clear-RecycleBin	Deletes the content of a computer's recycle bin
Restart-Computer	Restarts the operating system on local and remote computers
Restart-Service	Stops and then starts one or more services

Understand Windows PowerShell command syntax

Cmdlet structure

- The cmdlet’s verb is the action the cmdlet performs, such as:
  - Get
  - Set
  - New
  - Add
  - Remove
- The cmdlet’s noun is the resource the cmdlet affects, such as:
  - Service
  - Process
  - Use prefixes to group related nouns, including **AD**, **SP**, and **Az**

Parameters

- Parameters modify the action of a cmdlet.
- Names are entered starting with a dash (-).
- Parameters can be optional or required:
  - You’ll receive prompts for required parameters, if needed.
- Some accept multiple values, separated by commas.
- Parameter names are optional for positional parameters.

Tab completion

- Enables you to enter a few characters of a cmdlet or a parameter in the ISE or console and then press the Tab key on the keyboard.
- Allows you to enter cmdlet, parameter, variable, and path names more quickly and accurately.
- Helps you to discover cmdlets and parameters.
- Supports the use of wildcards.

About files

- Provide documentation for global shell techniques, concepts, and features.
- Start with **about_**.
- Review list by running **Get-Help about***.
- You’ll need to read many of these files to complete several upcoming lab exercises.

Demonstration: Using About files

Find commands and get help in Windows PowerShell

What are modules?

- Modules:
  - Are containers for related cmdlets.
  - Are provided as part of management tools for various software packages.
  - Must be loaded into your current session.
  - May only support specific operating systems.
- Windows PowerShell version 3.0 and newer support autoloading.
- Windows PowerShell and PowerShell Core support different module paths as indicated by the *$Env:PSModulePath* environment variable.

Finding cmdlets

- Use **Get-Command** and **Get-Help**, both of which support wildcards.
- Use **–Noun**, **–Verb**, and **–Module** parameters with **Get-Command**.
- **Get-Help** can also search help files if no match is found when searching command names.
- Use the **Find-Command** and the **PowerShellGet** module to find modules and commands from the PowerShell Gallery.

What are aliases?

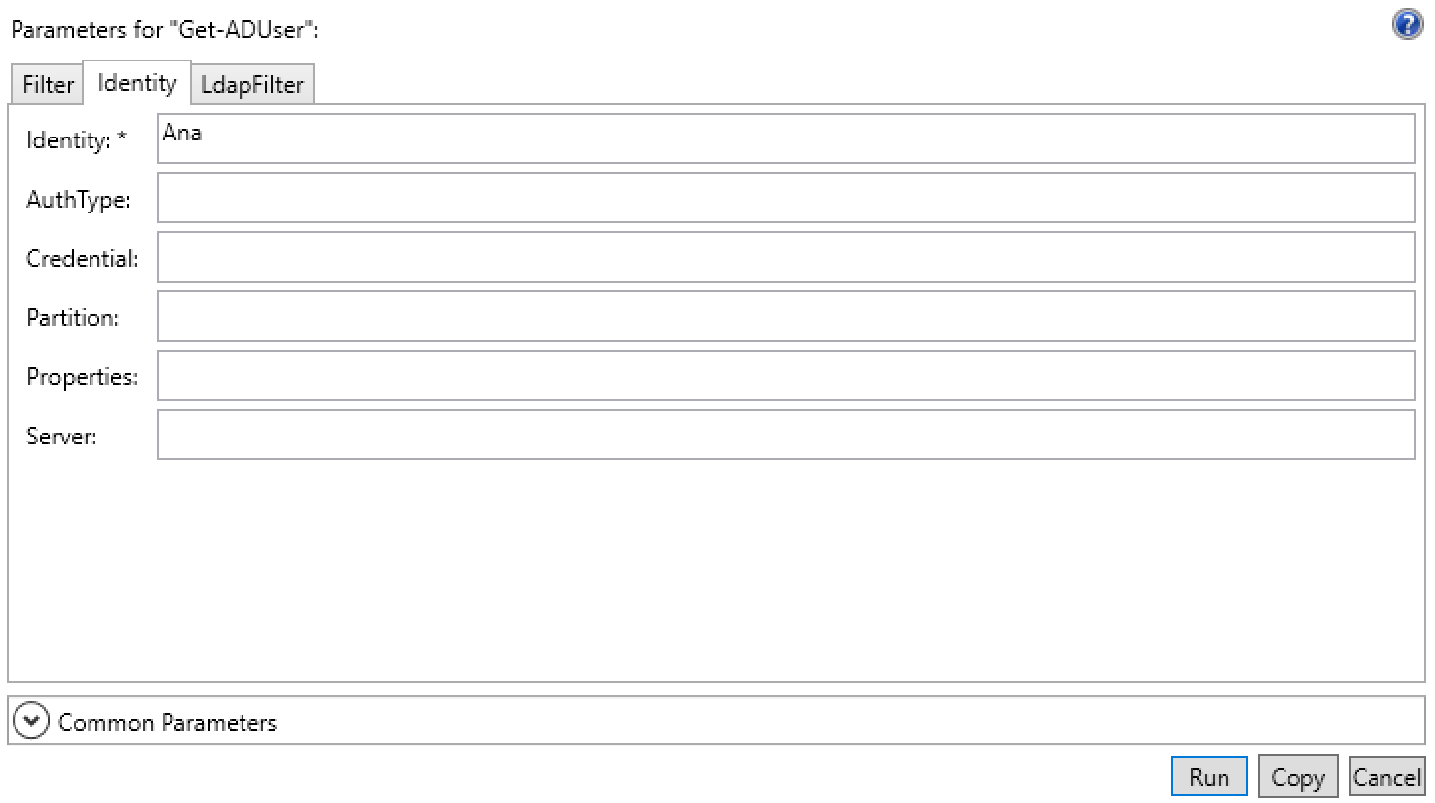
- Familiar batch commands include:
  - **Dir**
  - **Cd**
  - **Mkdir**
  - **Type**
- These are really aliases to Windows PowerShell commands.
- External commands such as **ping.exe** and **ipconfig.exe** all work as usual.
- Windows PowerShell commands often have a different syntax, even if accessed by an alias that matches an older command

Demonstration: Using aliases

- In this demonstration, you'll learn how to:

1. Find an alias for a cmdlet.
2. Find a cmdlet based on an alias you already know.
3. Create an alias.

Using Show-Command


Parameters for "Get-ADUser":

Identity: *	Ana
AuthType:	
Credential:	
Partition:	
Properties:	
Server:	

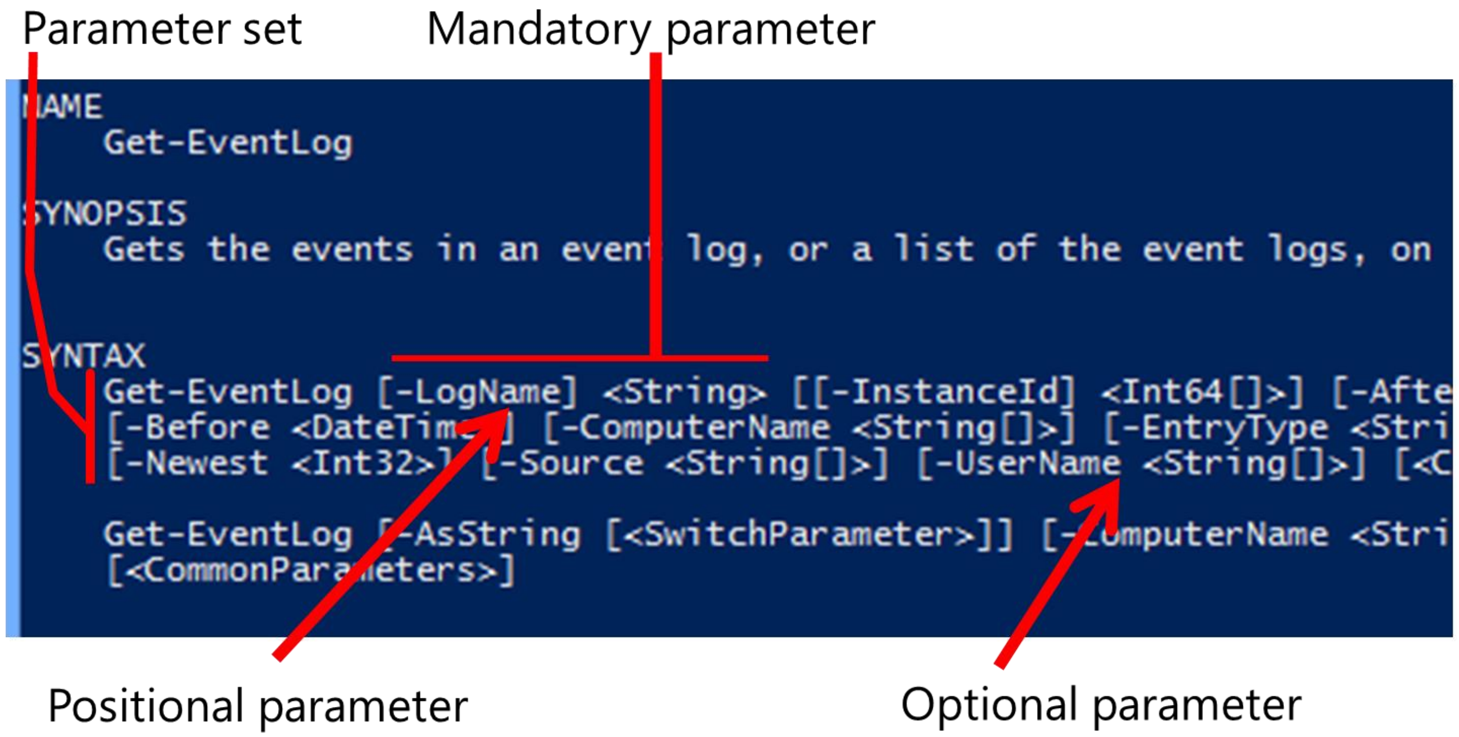
```
Get-ADUser -Identity Ana
```

Using Get-Help

- Displays Windows PowerShell help content.
- You provide a cmdlet name to display help for a cmdlet.
- Supports wildcards.
- Parameters include:
  - *-Examples*
  - *-Full*
  - *-Online*
  - *-ShowWindow*
  - *-Parameter ParameterName*

Demonstration: Reviewing Help

Interpreting the help syntax



```
NAME
    Get-EventLog

SYNOPSIS
    Gets the events in an event log, or a list of the event logs, on


SYNTAX
    Get-EventLog [-LogName] <String> [[-InstanceId] <Int64[]>] [-Afte
    [-Before <DateTime>] [-ComputerName <String[]>] [-EntryType <Stri
    [-Newest <Int32>] [-Source <String[]>] [-UserName <String[]>] [<C

    Get-EventLog [-AsString [<SwitchParameter>]] [-ComputerName <Stri
    [<CommonParameters>]
```

Updating help

- Windows PowerShell 3.0 and newer versions don’t ship with help files.
- **Update-Help**:
  - Uses downloadable help content to update your local help.
  - Checks no more than once every 24 hours by default.
- **Save-Help** enables you to download help and save it to an alternate location accessible to computers that aren’t connected to the internet.

Using background jobs and scheduled jobs

What are background jobs?

- Run commands in the background
- Store command results in memory for retrieval
- Three basic job types:
  - Local
  - Remoting
  - CIM/WMI
- Each job type has different characteristics

Starting jobs

- Local jobs:

```
Start-Job –ScriptBlock { Dir }
```

- Remoting jobs:

```
Invoke-Command –ScriptBlock { Get-Service } -ComputerName LON-DC1 –AsJob
```

- CIM/WMI jobs:

```
Start-Job  -ScriptBlock {Get-CimInstance –ClassName Win32_ComputerSystem}
Get-WmiObject –Class Win32_BIOS-ComputerName LON-DC1 -AsJob
```

Managing jobs

Use the following commands:

- **Get-Job**
  - Add –*ID* to retrieve a specific job by ID
  - Add –*Name* to retrieve a specific job by name
  - To get a list of child jobs, use the following syntax:

```
Get-Job –ID <parent_ID> -IncludeChildJobs
```

- **Stop-Job**
- **Remove-Job**
- **Wait-Job**

Retrieving results for running jobs

- Use **Receive-Job**:
  - Pipe jobs to it to specify jobs
  - Use –*ID* to specify by job ID
  - Use –*Name* to specify by job name
- Add –*Keep* to retain a copy of the results in memory. Otherwise, results aren’t retained in memory
- Receiving results from a parent job will result in receiving results from all child jobs

Demonstration: Using background jobs

In this demonstration, you'll learn how to create and manage local and remoting jobs.

1. Start local and remoting jobs.
2. Manage jobs.
3. Receive job results.

Running Windows PowerShell scripts as scheduled tasks

- Windows PowerShell scripts can run in **Task Scheduler**:
  - Can use all options of **Task Scheduler** in a graphical user interface
- A scheduled task consists of:
  - **Action**
  - **Principal**
  - **Trigger**
  - **Additional settings**
- To review the complete list of commands, run the following command:

```
Get-Command –Module ScheduledTasks
```

Demonstration: Using a Windows PowerShell script as a scheduled task

In this demonstration, you’ll learn how to create and run a Windows PowerShell script as a scheduled task.

What are scheduled jobs?

- Scheduled jobs are a cross between background jobs and **Task Scheduler** tasks
- They have three components:
  - Job definition
  - Job options
  - Job triggers
- To review all scheduled job commands, use:

```
Get-Command –Module PSScheduledJob
```

Job options and job triggers

- Use **New-ScheduledJobOption** to create an option object.
- Parameters correspond to options in the **Task Scheduler** GUI and include:
  - *–RequireNetwork*
  - *–RunElevated*
  - *–WakeToRun*
  - *–HideInTaskScheduler*
- Use **New-JobTrigger** to create a trigger object.
- Five basic types of triggers:
  - *–Once*
  - *–Weekly*
  - *–Daily*
  - *–AtLogOn*
  - *–AtStartUp*

Creating a scheduled job

- Use **Register-ScheduledJob** to create and register a new scheduled job.
- Windows PowerShell creates the job definition XML file on disk.
- Parameters include:
  - **–***Name*
  - *–ScriptBlock* or *–FilePath*
  - *–Credential*
  - *–MaxResultCount*
  - *–ScheduledJobOption* (job option object)
  - *–Trigger* (job trigger object)

Retrieving scheduled job results

- Run **Get-Job** to review a list of **PSScheduledJob** jobs:
  - Each represents a running of the scheduled job.
  - Provides access to the job results.
- Run **Receive-Job** to retrieve results.
- Run **Remove-Job** to delete results and the job object.

Demonstration: Using scheduled jobs

In this demonstration, you’ll learn how to create, run, and retrieve the results from a scheduled job.

Hvala na pažnji!