

# FUNDAMENTACIÓN CONCEPTUAL: AUTOMATIZACIÓN CON POWERSHELL



## 1. La Justificación Organizacional de PowerShell

PowerShell no es solo una "línea de comandos"; es un **marco de automatización y scripting basado en .NET** diseñado específicamente para la administración del sistema operativo Windows y sus servicios (incluyendo Active Directory, SQL Server, Exchange, etc.).

### 1.1 El Salto de CMD a PowerShell

Característica	CMD (Símbolo del Sistema)	PowerShell	Impacto Organizacional
Objeto de Salida	Texto (cadenas de caracteres).	Objetos .NET estructurados.	Permite <b>encadenar comandos</b> (pipelining) y realizar filtrado avanzado sin necesidad de procesar texto, logrando mayor precisión y eficiencia.
Capacidad de Scripting	Batch (.bat) — Limitado y secuencial.	Scripts (.ps1) — Completo, con lógica, bucles y funciones avanzadas.	<b>Escalabilidad:</b> Las tareas se automatizan por completo. La creación de 500 usuarios se realiza con un único <i>script</i> en lugar de 500 comandos manuales.
Descubrimiento	Nulo; requiere conocimiento previo del comando.	Sistema de <b>Verbo-Nombre y Get-Command</b> .	<b>Eficiencia del Administrador:</b> Permite a los técnicos descubrir nuevas funcionalidades y comandos de forma lógica y rápida.

Exportar a Hojas de cálculo

### 1.2 La Necesidad de la Automatización

En cualquier organización con más de diez usuarios, la administración manual se vuelve ineficiente e inconsistente. PowerShell resuelve esto:

- **Consistencia:** Un script de aprovisionamiento de un nuevo empleado (`New-User.ps1`) garantiza que todos los usuarios tengan las mismas propiedades, el mismo formato de nombre de cuenta y la misma ubicación de OU.
  - **Gestión Remota y Server Core:** Permite administrar servidores sin interfaz gráfica (**Server Core**) y ejecutar comandos en docenas de servidores simultáneamente, reduciendo la necesidad de acceso físico o sesiones RDP.
- 

## 2. Estructura y Sintaxis Fundamental

PowerShell opera bajo principios muy estrictos que facilitan su uso y aprendizaje.

### 2.1 La Estructura Verbo-Nombre (Cmdlets)

Todo comando en PowerShell (llamado **Cmdlet**) sigue el estándar **Verbo-Nombre** (ej., `Get-ADUser`). Esto facilita la **descubridad** de comandos:

Verbo	Propósito	Ejemplo
<code>Get</code>	Recuperar o leer datos (lectura).	<code>Get-Service, Get-ADGroup</code>
<code>Set</code>	Modificar propiedades existentes.	<code>Set-ADUser, Set-ExecutionPolicy</code>
<code>New</code>	Crear un recurso o instancia.	<code>New-ADUser, New-Item</code>
<code>Add / Remove</code>	Agregar o eliminar elementos de una colección.	<code>Add-ADGroupMember, Remove-Item</code>

Exportar a Hojas de cálculo

El comando fundamental para aprender PowerShell es `Get-Command` y `Get-Help`, ya que permiten al administrador descubrir y entender cualquier funcionalidad disponible.

### 2.2 El Concepto del Pipeline (|) y Objetos

A diferencia de los *pipes* de CMD (que pasan texto), el *pipeline* (|) de PowerShell pasa **objetos .NET** de un cmdlet a otro.

**Objeto:** Una estructura de datos que contiene propiedades (atributos) y métodos (acciones).

**Ejemplo de Objeto:** El comando `Get-Service` no devuelve texto, sino objetos de tipo `ServiceController`. Cada objeto tiene propiedades como `Status`, `Name`, y `DisplayName`.

PowerShell

```
# Ejemplo: Filtrar los servicios que están detenidos y seleccionar solo dos propiedades  
Get-Service | Where-Object { $_.Status -eq "Stopped" } | Select-Object Name, DisplayName
```

- **| (Pipeline)**: Pasa la salida de `Get-Service` al siguiente cmdlet.
- **Where-Object**: Filtra los objetos basándose en la propiedad `Status`.
- **Select-Object**: Muestra solo las propiedades deseadas (Nombre y Nombre para mostrar).

Este encadenamiento permite realizar tareas complejas en una única línea de código.

## 2.3 Módulos y Variables

- **Módulos**: Son bibliotecas de *cmdlets* que deben cargarse para ser utilizados. Ejemplos críticos son `ActiveDirectory`, `DHCPServer`, y `FailoverClusters`. La mayoría se cargan automáticamente (`Import-Module`).
- **Variables**: Se definen con el símbolo `$` (ej., `$Miembros`, `$Fecha`). Las variables almacenan objetos, lo que permite reutilizar el resultado de un comando o construir *scripts* complejos.

---

## 3. Trabajo Práctico: Automatización de Administración (Clases 5 y 6)

El enfoque práctico se centra en el módulo `ActiveDirectory` para aplicar los principios de identidad y seguridad vistos en las clases anteriores.

### 3.1 Gestión Masiva y Creación (CRUD)

La gestión de **CRUD (Create, Read, Update, Delete)** es la aplicación más común de PowerShell en AD.

Tarea	Cmdlet	Fundamentación
<b>Crear Usuario</b>	<code>New-ADUser</code>	Asegura la <b>consistencia</b> en la sintaxis de nombres de cuenta ( <code>sAMAccountName</code> ) y la ubicación de la OU ( <code>-Path</code> ).
<b>Obtener Propiedades</b>	<code>Get-ADUser</code>	Permite la <b>auditoría</b> y la <b>lectura masiva</b> de atributos de identidad (ej., obtener todos los usuarios con el atributo "Departamento" igual a "Ventas").

<b>Modificar Propiedades</b>	<code>Set-ADUser</code>	Permite la <b>actualización masiva</b> de atributos (ej., cambiar el atributo "Compañía" para todos los usuarios de una OU).
<b>Gestión de Grupos</b>	<code>Add-ADGroup Member</code>	Esencial para el modelo <b>AGDLP</b> . Permite la <b>automatización</b> de la asignación de roles.

Exportar a Hojas de cálculo

### 3.2 El Poder del Filtro (**-Filter**) y la Exportación

La eficiencia de PowerShell reside en su capacidad para filtrar grandes conjuntos de datos de AD sin sobrecargar el Controlador de Dominio (DC).

- El parámetro **-Filter** utiliza la sintaxis de filtro de LDAP de forma optimizada.

PowerShell

```
# Ejemplo: Filtrar usuarios inactivos y exportar un informe de auditoría.
# (LastLogonDate es una propiedad replicada de forma imperfecta; LastLogonTimeStamp
# es más fiable para grandes búsquedas).
```

```
# 1. Obtener usuarios que NO han iniciado sesión en más de 90 días
Get-ADUser -Filter { LastLogonDate -le (Get-Date).AddDays(-90) } -Properties
LastLogonDate, Enabled |
```

```
# 2. Seleccionar solo las propiedades relevantes para el informe
Select-Object Name, LastLogonDate, Enabled, DistinguishedName |
```

```
# 3. Exportar a un archivo CSV (CSV es el estándar para informes de auditoría)
Export-Csv "C:\Temp\Usuarios_Inactivos.csv" -NoTypeInformation
```

Este ejercicio práctico demuestra cómo PowerShell transforma una tarea manual tediosa y propensa a errores (auditar manualmente 1000 usuarios) en un **proceso automatizado, auditible y documentado**.