1 PowerShell

1.1 Sumario

1.2 Introdución

Windows PowerShell é unha interfaz de consola (CLI) con posibilidade de escritura e unión de comandos por medio de instruccións (*scripts*). É moito máis rica e interactiva que os seus predecesores, dende DOS ata Windows 7. Esta interfaz de consola está deseñada para o seu uso por parte de administradores de sistemas, có propósito de automatizar tarefas ou realizalas de forma máis controlada. Orixinalmente denominada como **MONAD** en 2003, o seu nome oficial cambiou ao actual cando foi lanzada ao público o 25 de abril de 2006.

1.2.1 Versións de Powershell

- **PowerShell 1.0**: PowerShell 1.0 foi liberado en 2006 para o Windows XP SP2, Windows Server 2003 e Windows Vista. É un compoñente opcional para Windows Server 2008.

- **PowerShell 2.0**: PowerShell 2.0 está integrado có Windows 7 e Windows Server 2008 R2 e é liberado para Windows XP con Service Pack 3, Windows Server 2003 con Service Pack 2 e Windows Vista con Service Pack 1.

  - PowerShell V2 inclúe cambios da linguaxe de scripting e a API do equipo, engadindo uns 240 cmdlets.
  - **Windows PowerShell ISE v2.0**, é unha contorna de desenvolvemento para scripts PowerShell scripts PowerShell 2.0, está integrado no Windows 7 e no Windows Server 2008 R2 e é liberado para Windows XP có Service Pack 3, Windows Server 2003 có Service Pack 2 e Windows Vista có Service Pack 1.

- **PowerShell 3.0**: PowerShell 3.0 está integrado no Windows 8 e no Windows Server 2012. Microsoft tamén fixo PowerShell 3.0 posible para Windows 7 có Service Pack 1, para Windows Server 2008 có Service Pack 1 e para Windows Server 2008 R2 có Service Pack 1.

  - PowerShell 3.0 é unha parte dun paquete moito máis grande, ***Windows Management Framework* 3.0 (WMF3)**, que tamén contén o servizo WinRM.

- **PowerShell 4.0**: PowerShell 4.0 está integrado con Windows 8.1 e con Windows Server 2012 R2. Microsoft tamén fixo posible PowerShell 4.0 para Windows 7 SP1, Windows Server 2008 R2 SP1 e Windows Server 2012.

- **PowerShell 5.0**: A revisión previa do PowerShell 5.0 estivo dispoñible con Windows Management Framework 5.0 (WMF5) en April de 2014.

  - Unha das novedades que introduce é OneGet PowerShell cmdlets para soportar o manexo de paquetes do repositorio Chocolatey.
  - En Agosto de 2015 Microsoft libera a última actualización de PowerShell 5.0.
  - Windows 10 xa trae integrado

- **PowerShell 5.1**: Aparece coa Versión 1607 de Windows 10 (Anniversary Update) e en Windows 2016.

- **PowerShell Core 6.0, 6.1 y 6.2**: Anunciada en Agosto do 2016.

- PowerShell Core é unha PowerShell máis pequena que a PowerShell convencional e pode instalarse en sistemas operativos: MacOS e Linux, tamén en Windows.
Con PowerShell Core, poderemos administrar e executar os mesmos scripts sobre os tres tipos de sistemas operativos: Windows MacOS e Linux como se foran un só.

- **PowerShell 7**: Será a próxima versión de PowerShell.

- Remprazará o PowerShell Core 6.x e a Windows PowerShell 5.1.

1.2.2 Sistemas Operativos e Versións de Powershell

Os Sistemas Operativos Windows e Windows Server están actualizando as súas versións de PowerShell:

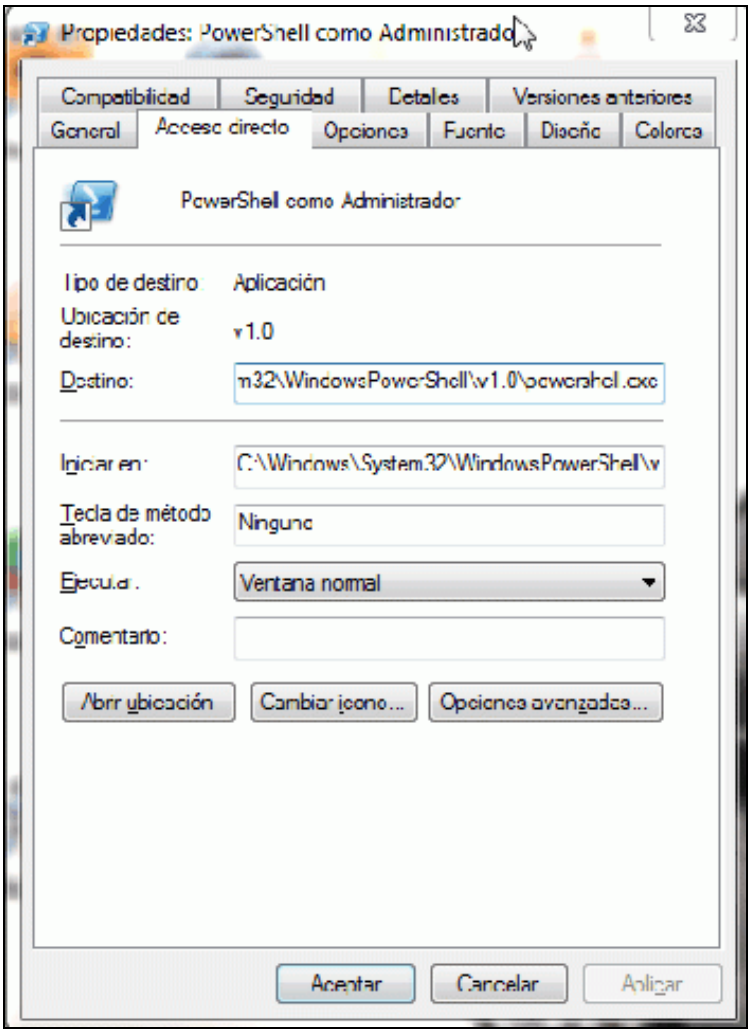
- **Windows PowerShell 4.0** podemos atopalo no Windows 10, no Windows 8.1 e no Windows Server 2012 Release 2 (R2). Tamén é posible instalalo en equipos con Windows 7 with Service Pack 1 ou posteriores e Windows Server 2008 R2 con Service Pack 1 ou posteriores coa instalación do Windows Management Framework 4.0.
- **Windows PowerShell 3.0** podemos atopalo no Windows 8 e no Windows Server 2012. Tamén podemos instalalo en ordenadores que teñan Windows 7 có Service Pack 1 ou posteriores, Windows Server 2008 R2 có Service Pack 1 ou posteriores e Windows Server 2008 có Service Pack 2 ou posteriores coa instalación do Windows Management Framework 3.0.

1.2.3 Executar Windows PowerShell

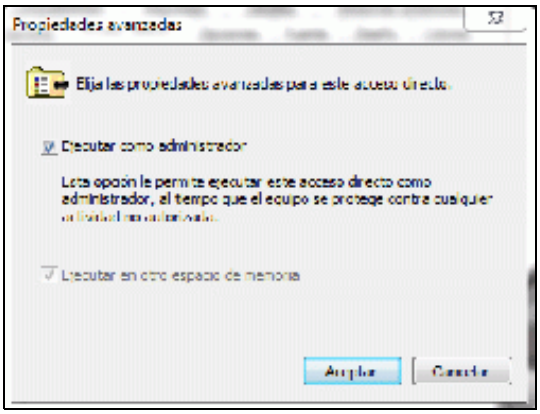
Pulsar **Tecla de Windows + R** para que se execute "Executar", escribir ***PowerShell*** e pulsar ***Enter***.

◊ **Executar Windows PowerShell como administrador**:

1. Crear un acceso directo no escritorio. Onde pon "Escriba a localización do elemento" escribimos simplemente "PowerShell" e pulsamos en "Seguinte".
2. Logo dámoslle un nome para este acceso directo como, por exemplo: "PowerShell como Administrador".
3. Para que se execute como Administrador, unha vez creado o acceso directo, facemos click sobre el có botón dereito e na ficha "Acceso directo" pulsamos no botón "Opcións avanzadas...".



Aparece unha nova fiestra na que temos que seleccionar a casiña de verificación "Executar como administrador", logo pulsamos en aceptar e xa temos preparado o acceso director para executar en calquera momento o PowerShell con privilexios de Administrador.



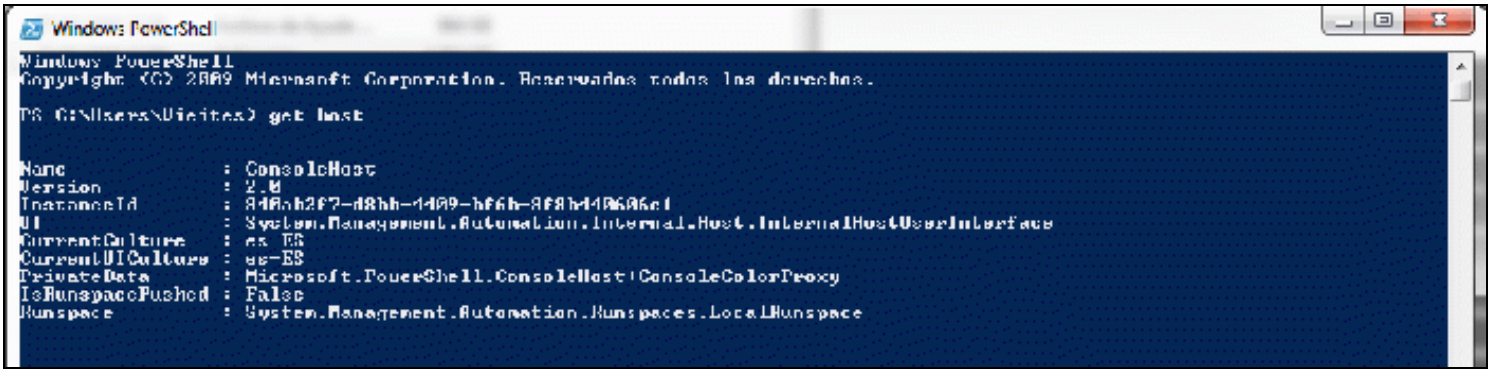
◊ **Executar Powershell como administrador dende liña de comandos:"**

Para realizar esta tarefa emprégase o cmdlet Start-Process.

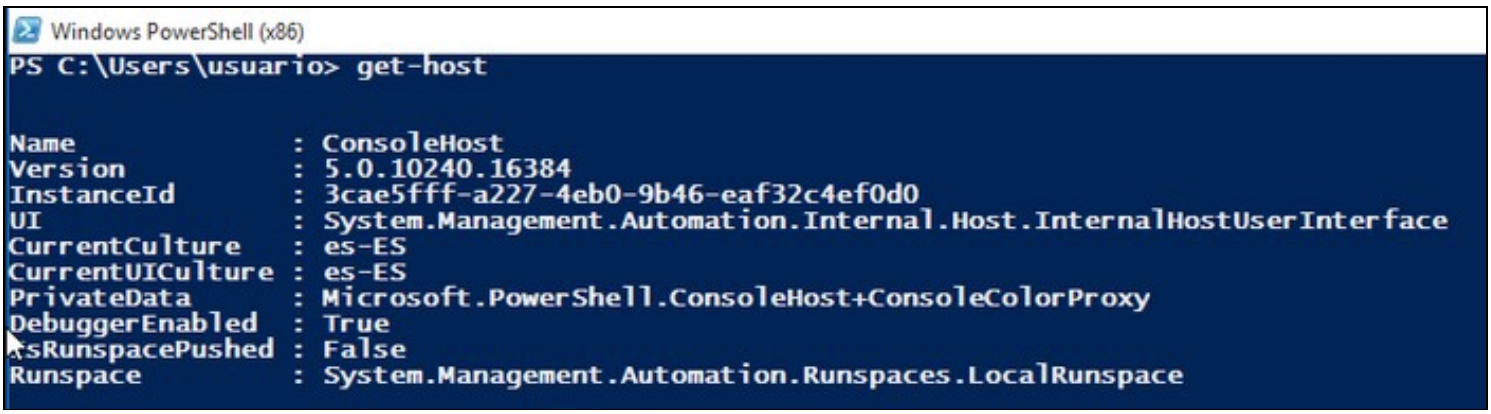
```
# Abrir powershell como administrador:
PS> Start-Process Powershell -Verb RunAs

# Executar un script powershell como administrador:
PS> Start-Process Powershell -Verb RunAs -ArgumentList "-file c:\scripts\cambionome.ps1"
```

1.2.4 Coñecer a versión de PowerShell instalada no noso equipo

Para coñecer a versión instalada no noso equipo empregamos o comando: **get-host**



Nesta imaxe vemos que a versión é a 2.0 (trátase dun Windows Server 2008).



Nesta outra imaxe vemos que nun Windows 10 xa se atopa instalado o Powershell 5.0.

1.2.5 Configurar o equipo para executar scripts PowerShell

Antes de nada, debemos saber que:

- Os **scripts** de PowerShell son arquivos de texto gardados con extensión .PS1.
- Para crealos podemos empregar o propio Notepad ou, mellor, o software **PowerShell ISE** que se atopa xa instalado nos sistemas operativos Windows actuais. Tamén existen tamén ferramentas de terceiros como **PowerGUI**.
- Por defecto, para mellorar a seguridade, estes arquivos os abre o **Notepad** polo que, se lle decimos en **Autologon** que se executen o único que ocorre é que se "abren" sen mais (vese o contido). Se nos interesa, podemos configurar o Sistema Operativo Windows para que os arquivos con extensión **.ps1** se executen todos por defecto có programa **Powershell.exe**.
- Ademais, moitos deles, hai que executalos con permisos de Administrador (hai que baixar o nivel de ?Notificación acerca de cambios en el equipo?, o **UAC**):

  - Baixar UAC.

- Por último, hai que cambiar a preferencia do usuario para a directiva de execución de Windows PowerShell, executando o comando:

```
PS>Set-ExecutionPolicy RemoteSigned
```

Para ver se cambiou o nivel executas o comando:

```
PS>Get-ExecutionPolicy
```

- Pódese axustar como o PS se comporta cando se produce un erro cando se executa código ou scripts. Para ver o modo actual empregaremos a variable ErrorActionPreference:

```
PS>$ErrorActionPreference
```

```
Continue
```

O modo por defecto vemos que é **Continue** que mostra os erros e continúa coa execución. Se queremos que non mostre os erros e tira para adiante teríamos que configuralo en **SilentlyContinue**.

```
PS>$ErrorActionPreference = "SilentlyContinue"
```

1.2.6 Mini-Introdución de manexo de PowerShell

- Executar Powershell:

  No casiña de "buscar" de Windows teclear "PowerShell" e pulsar **Intro**.

- Executar Powershell como administrador:

```
PS> Start-Process Powershell -Verb RunAs
```

- Executar **Powershell ISE** para a creación de *scripts*.
- Comprobar a versión de Powershell que estamos utilizando:

```
PS> get-host
```

- Configurar o equipo para executar *scripts* de PS realizados por nosotros e baixados de Internet (firmados):

```
PS> Get-ExecutionPolicy
# Restricted
# AllSigned
# RemoteSigned
# Unrestricted
PS> Set-ExecutionPolicy RemoteSigned
```

- Los comandos de PS son denominados ahora *cmdlets*.
- Exemplo típico cun *cmdlet* como get-date:

```
PS> get-date
PS> get-help get-date
PS> get-date | fl
PS> get-date | fl day, hour, minute, second
PS> $fecha = get-date
PS> $fecha.gettype()
PS> $fecha.gettype().name
PS> $fecha
PS> $fecha | fl
PS> $fecha | get-member
PS> $fecha | get-member -MemberType Property
PS> $fecha.date
PS> $fecha | get-member -MemberType Method
PS> $fecha.date
PS> $fecha.AddDays(10)
```

- Ver de que comandos dispoñemos para traballar:

```
PS> get-command -noun *date*
PS> Get-Command -CommandType cmdlet | Measure-Object -Line
PS> Get-Command -verb get | more
PS> Get-alias -Name rm
PS> get-alias | where {$_.Name -like "*m*"}
```

- Administración do sistema:

```
# Apagar o equipo
PS> Stop-Computer
# Reiniciar o equipo
PS> Restart-computer
```

- Información do sistema:

```
PS> DIR env:    # Directorio de variables de contorna
PS> $env:computername    # Nome de equipo
PS> $env:username    # Usuario que iniciou sesión
PS> Get-WmiObject -Class Win32_Computersystem    # Descubrir a configuración do equipo
```

1.3 Cmdlets: entrada/saída básica

A diferencia da maioría das ferramentas de liñas de comandos, que aceptan e devolven texto, Windows PowerShell baséase en **Microsoft .NET Framework** e acepta e devolve obxectos de **.NET Framework**. Este cambio fundamental no contorno aporta ferramentas e métodos totalmente novos que melloran en grande medida o control, a eficiencia e a productividade de programadores e administradores.

Windows PowerShell é unha ferramenta de liña de comandos simple que introduce o concepto de **cmdlet**. Un **cmdlet** é unha combinación de verbo e nome que comprende un comando e un obxecto sobre o que actúa o comando. Os nomes de *cmdlet* de Windows PowerShell constan de verbos e nomes, separados por un guión (-), que en conxunto denotan as súas propiedades funcionáis. Por exemplo, o nome de *cmdlet* **Get-Date** combina o verbo (comando) "Get" có nombe (obxecto) "Date" para denominar ao *cmdlet* que recupera un obxecto *Date* especificado. Pode usar os *cmdlets* individualmente ou unilos como secuencias vinculadas para realizar tarefas complexas.

Os nomes de *cmdlet* toman parámetros como pares nome-valor que otorgan especificidade ao nome de *cmdlet*. Cando se invocan, os *cmdlets* devolven obxectos de saída. Estes obxectos devoltos tamén teñen propiedades que se mostran como pares nome-valor. Debido a que os *cmdlets* devolven obxectos, estes últimos poden pasarse (ou "canalizarse") a outro *cmdlet*, en secuencia. Deste xeito, os *cmdlets* poden encadenarse e proporcionar unha grande flexibilidade.

É importante ter en conta que un *cmdlet* non é un executable, senón que é unha sesión dunha clase de .NET Framework. Polo tanto, cunhas poucas excepcións, os *cmdlets* devolven obxectos en lugar de secuencias de texto e procesan os obxectos de entrada dunha canalización de obxectos.

No seguinte enlace vemos unha colección de **cmdlets** de mantemento moi interesantes e no seguinte unha colección de utilidades *cmdlets*.

Como pode verse, **Windows PowerShell** non é só unha ferramenta de liña de comandos. Tamén é, máis importante ainda, unha nova linguaxe de *scripting*. Por exemplo, Windows PowerShell se instala de forma nativa en Windows 10 con **496 *cmdlets***. Outro exemplo é que Windows Server 2012 promocionado a Servidor de Dominio e no que tamén está instalado o servizo DNS, ten **656 *cmdlets***.

No seguinte enlace podemos atopar axuda sobre os elmentos do núcleo de **PowerShell**.

- **Por exemplo, se queres saber a data/hora actual:**

```
PS>get-date
miércoles, 19 de diciembre de 2012 14:50:30
```

Se queres ver a súa saída en formato **lista** (*Format-List*):

```
PS>get-date | fl
...
```

Se queres ver a súa saída en formato **táboa** (*Format-Table*):

```
PS>get-date | ft
...
```

Para ver todos os **métodos** e **propiedades** da clase *get-date* podemos redirixir a súa saída ao *cmdlet* **get-member**:

```
PS>get-date | get-member
...
```

Así, para ler algunha desas propiedades debemos facer o seguinte:

```
PS>$data = get-date
PS>$data.year
2012
```

Por último, para ver a data en formato epoch:

- Web moi interesante onde podemos ver como traballar có formato epoch.

```
PS>$data = get-date -UFormat "%s"
PS>$data
1385854825,56271
```

- Para mostrar por pantalla mensaxes emprégase o *cmdlet* **Write-Host**:

```
PS>Write-Host Hola, estamos no ano $data.year
Hola, estamos no ano 2012
#
# Para concatenar mellor facer así:
PS> Write-Host "Hola, estamos no ano $($data.year)"
Hola, estamos no ano 2012
```

- Para pedir o valor dun parámetro ao usuario pode empregarse o *cmdlet* **Read-Host**:

```
PS>$nome = Read-Host "Introduce o teu nome: "
Introduce o teu nome: Manuel
PS> Write-Host "Hola $nome"
Hola Manuel
# En PowerShell o carácter de escape é "`" a comiña invertida ou ''backtick''
PS> Write-Host "Hola `$nome"
Hola $nome
```

Tamén podes poñelo todo nunha mesma liña:

```
PS>$nome = Read-Host "Introduce o teu nome: "; Write-Host "Hola $nome"
Introduce o teu nome: Manuel
Hola Manuel
```

- Para ver unha lista de todos os *cmdlets*, *aliases*, *functions*, *workflows*, *filters*, *scripts* e *applications* dispoñibles utiliza: Get-Command.

Se queremos ver só os *cmdlets*:

```
#Ver a lista dos cmdlets:
PS>Get-Command -commandType cmdlet
...
#Contar o número de cmdlets existentes:
PS>Get-Command -commandType cmdlet | Measure-Object -Line
Lines ...
---
496
#Ver os cmdlets que realizan a acción "get":
PS>Get-Command -verb get
...
```

- Para acceder á axuda utiliza Get-Help.

```
#Ver a axuda do cmdlet Get-Command:
PS>Get-Help Get-Command -detailed
...
#Descubrir comandos que realicen certas accións
#empregando "comodíns":
PS>Get-Command *help* -CommandType cmdlet
CommandType     Name        Definition
-----------     ----        ----------
cmdlet          Get-Help    Get-Help [[-Name] <String>] [-Path <String>] [-C...
```

1.3.1 Agregar un novo módulo con máis cmdlets

Podemos agregar un novo módulo que nos permita ter novos cmdlets. Vexamos un exemplo con cmdlets que nos permiten actualizar o sistema operativo. Trátase do módulo PSWindowsUpdate.
Para instalalo hai que descargar o zip e descomprimilo nos directorios:

%USERPROFILE%\Documents\WindowsPowerShell\Modules
%WINDIR%\System32\WindowsPowerShell\v1.0\Modules

1.4 Execución de *scripts*

PowerShell executa *scripts* con extensión **.ps1**, de xeito que podes escribir todo o código nun arquivo de texto coa extensión indicada.

Para editar os *scripts* pódese utilizar un editor de texto calquera ou, mellor, a aplicación **PowerShell ISE** xa instalada nos Windows actuais. Tamén existen outras aplicacións como PowerGui, xa antes mencionada.

Para executar un *script* hai que indicar a súa dirección completa, isto podemos facelo de dous xeitos:

- Dende a liña de comandos convencional:

```
powershell ./hola.ps1
powershell c:\scripts\hola.ps1
```

- Dende a liña de comandos PowerShell:

```
PS>./hola.ps1
PS>c:\scripts\hola.ps1
```

1.5 Comentarios

Os comentarios en PowerShell escríbense empregando o símbolo almofada (**#**). Todo o que se atope despois deste carácter, non será interpretado polo PowerShell.

```
#Este é un comentario.
<#

 Para
 comentarios
 multilínea

 #>
```

1.6 Tipos de datos: variables, *arrays*, listas e táboas *hash*

Neste apartado veremos como definir e traballar cós tipos de datos máis empregados en PowerShell.

1.6.1 Variables

As variables comezan por **$** e podemos meter nelas casi calquera ?cousa?, por exemplo:

```
#Obxecto DateTime
PS>$data = Get-Date
PS>$cadea.GetType().Name  #Ver o tipo de variable que é
DateTime

#Enteiro
PS>$numero = 1
PS>$numero.GetType().Name
Int32

#Decimal
PS>$numero_decimal = 6.5
PS>$numero_decimal.GetType().Name
Double

#String dunha única letra
PS>$letra = 'p'
PS>$letra.GetType().Name
String

#String de máis dun carácter
PS>$cadea = 'isto é unha cadea'
PS>$cadea.GetType().Name
String
```

Para crear unha cadea de texto podemos facer referencia a outra variable:

```
PS>$result = 1.50
PS>$texto = ?O pan vale $result?
```

Para acceder ao contido da variable só hai que poñer o seu nome e, como vimos agora, para crear un texto có contido de outra variable só temos que poñer esta entre comiñas dobres. Tede en conta que non se pode facer isto coas comiñas simples, pois estas non resolven variables.

```
PS>$result = 1.50
PS>$texto = ?O pan vale $result?
PS>$texto
O pan vale 1.50
```

Podes sumar dúas variables numéricas ou unir dúas variables de texto:

```
PS>$na = 1.50
PS>$nb= 3
PS>"$na + $nb = $($na + $nb)"
1.5 + 3 = 4,5
PS>$ta = "Ola"
PS>$tb = "amigo"
PS>$ta + $tb
Ola amigo
```

Nunha variable tamén podemos gardar temporalmente a saída dun comandou ou dun *cmdlet*:

```
PS>$contido = Get-ChildItem c:\
PS>$contido

    Directorio: C:\

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        19/12/2012      1:28            inetpub
d----        14/07/2009      5:20            PerfLogs
d-r--        19/12/2012      1:28            Program
d-r--        19/12/2012      1:29            Program
d----        18/12/2012      0:24            scripts
d-r--        05/10/2011     15:05            Users
d----        19/12/2012      1:28            Windows

#Ver o tipo de elemento que é $contido
PS>$contido.GetType().name
Object[]

#Ver os métodos, propiedades, etc do obxecto $contido
PS>$contido | Get-Member
...
```

Nota: Debes ter en conta que en PowerShell as variables NON son sensibles a maiúsculas/minúsculas.

1.6.1.1 Atopar variables

Tamén é moi interesante saber que podemos atopar as variables xa definidas empregando a unidade virtual "**variable:**", vexamos algún exemplo:

```
#Para mostrar todas as existentes
PS>Dir variable:
...

#Para mostrar todas as que comecen polas letras "nom":
PS>Dir variable:nom*
Name       Value
----       -----
nome2      Pepe
nome1      Xan

#Poñendo máis condicións:
PS>Dir variable:nom* -exclude *1*
Name       Value
----       -----
```

```
nome2      Pepe

#Para buscar unha variable polo valor que contén:
PS>dir variable: | Out-String -stream | Select-String " pepe "
nome2           pepe
```

Neste último exemplo a saída de **Dir** é pasada a Out-String que convirte o resultado nun texto. O parámetro **stream** asegura que toda variable subministrada por **Dir** é separada unha a unha como texto. O *cmdlet* Select-String selecciona as liñas que inclúen o valor desexado, filtrando o resto. Para asegurarse que o valor da variable é "pepe" e non unha palabra que conteña ese conxunto de caracteres, introducimos uns espazos en branco antes e despois do buscado.

1.6.1.2 Verificar que unha variable existe

Se queremos verificar se unha variable existe podemos empregar o *cmdlet* Test-Path. Podemos ver uns exemplos:

```
PS>$nome = "Andrea"
PS>Test-Path variable:\nome
True
PS>Test-Path variable:\apelido
False
```

1.6.1.3 cmdlets específicos para traballar con variables

Existen cinco *cmdlets* específicos para traballar con variables: Para conocelos podemos empregar o comando:

```
PS> Get-Command -Noun variable
CommandType     Name                    ModuleName
-----------     ----                    ----------
Cmdlet          Clear-Variable          Microsoft.PowerShell.Utility
Cmdlet          Get-Variable            Microsoft.PowerShell.Utility
Cmdlet          New-Variable            Microsoft.PowerShell.Utility
Cmdlet          Remove-Variable         Microsoft.PowerShell.Utility
Cmdlet          Set-Variable            Microsoft.PowerShell.Utility
```

- Clear-Variable : Limpa o contido da variable, polo que o contido desta pasa a ser **NULL**.
- Get-Variable : Obter o obxecto variable en si, non o valor da variable.
- New-Variable : Crear unha nova variable.
- Remove-Variable : Borra unha variable e o seu contido, sempre e cando esta non sexa unha constante ou unha variable creada polo sistema.

  Tamén podemos borrar unha variable como se tratase dun sistema de arquivos:

```
PS>$nome = "Andrea"
PS>del variable:\nome
```

- Set-Variable : Dalle un valor determinado a unha variable. Se a variable non existe créase.

1.6.1.4 Variable con descrición

As variables poden ter unha descrición opcional que nos axuda a atopala cando sexa preciso.

```
PS>New-Variable dato -value 100 -description "variable de proba" -force
```

Por outro lado, se mostramos esta variable vemos que esa propiedade aparece invisible a non ser que forcemos que o PowerShell a mostre:

```
PS>$mivariable
100
PS>Dir variable:\mivariable
 Name           Value
 ----           -----
 mivariable     100

#Para mostrar a descrición:
PS>Dir variable:\mivariable | Format-Table Name, Value, Description -autosize
Name        Value  Description
----        -----  -----------
mivariable   100  variable de proba
```

```
#Tamén podemos aproveitar esa propiedade para buscar a variable:
PS> dir variable: | Format-Table Name, Value, Description | Out-String -stream | Select-String "proba"
mivariable         100              variable de proba
```

1.6.1.5 Definindo constantes (variables protexidas)

Unha constante gardará un valor que non pode ser modificado. Traballan igual que as varibles pero con protección frente a escritura.

PowerShell non distingue entre unha variable e unha constane, pero nos ofrece seguridade frente a escritura. Vexamos como crealas:

```
PS> New-Variable constante -value 100 -option ReadOnly
PS> $constante
100
PS> $constante = 200
No se puede sobrescribir la variable constante porque es de solo lectura o constante.
...
```

Para cambiar o seu valor o mellor é eliminala e volver a creala, pero se intentamos iso precisaremos empregar a propiedade "**-force**":

```
PS> del variable:\constante -force
PS> $constante
PS> $constante = "abc"
PS> $constante
abc
```

1.6.1.6 Variables "Automáticas"

As "Variables Automáticas" son as que emprega PowerShell para o seu uso interno. Almacénanse no dispositivo **variable:**.

```
PS> Dir variable: | Sort-Object Name | Format-Table Name, Description -autosize

Name                                Description
----                                -----------
$
?                                   Estado de ejecución del último comando.
^
_
args
ConfirmPreference                   Indica cuándo debe solicitarse confirma
                                    ción. Se solicita confirmación cuando e
                                    l valor de ConfirmImpact es igual o may
                                    or que el de $ConfirmPreference. Si $Co
                                    nfirmPreference es None, sólo se solici
                                    tará confirmación cuando se especifique
                                     el valor Confirm.
ConsoleFileName                     Nombre del archivo de consola actual.
DebugPreference                     Indica la acción tomada cuando se entre
                                    ga un mensaje de depuración.
Error
ErrorActionPreference               Indica la acción tomada cuando se entre
                                    ga un mensaje de error.
ErrorView                           Indica el modo de vista que se debe usa
                                    r para mostrar los errores.
ExecutionContext                    Objetos de ejecución disponibles para l
                                    os cmdlets.
false                               Boolean False
FormatEnumerationLimit              Indica el límite de la enumeración al a
                                    plicar formato a los objetos de IEnumer
                                    able.
HOME                                Carpeta que contiene el perfil del usua
                                    rio actual.
Host                                Se trata de una referencia al host de e
                                    ste Runspace.
input
MaximumAliasCount                   Número máximo de alias permitidos en un
                                    a sesión.
MaximumDriveCount                   Número máximo de unidades permitidas en
                                     una sesión.
MaximumErrorCount                   Número máximo de errores para conservar
                                     en una sesión.
```

MaximumFunctionCount	Número máximo de funciones permitidas en una sesión.
MaximumHistoryCount	Número máximo de objetos de historial para conservar en una sesión.
MaximumVariableCount	Número máximo de variables permitidas en una sesión.
mivariable	variable de proba
MyInvocation	
NestedPromptLevel	Indica el tipo de confirmación que se debe mostrar para el nivel de anidación actual.
null	Las referencias a variables nulas siempre devuelven el valor nulo. Las asignaciones no surten ningún efecto.
OutputEncoding	La codificación de texto usada al canalizar texto hacia un ejecutable nativo.
PID	Id. de proceso actual.
PROFILE	
ProgressPreference	Indica la acción tomada cuando se entregan registros de progreso.
PSBoundParameters	
PSCulture	Referencia cultural de la sesión de Windows PowerShell actual.
PSEmailServer	Variable para albergar el servidor de correo electrónico. Se puede usar en lugar del parámetro HostName en el cmdlet Send-MailMessage.
PSHOME	Carpeta primaria de la aplicación host de este espacio Runspace.
PSSessionApplicationName	AppName donde se va a establecer la conexión remota.
PSSessionConfigurationName	Nombre de la configuración de sesión que se cargará en el equipo remoto.
PSSessionOption	Opciones de sesión predeterminadas para las nuevas sesiones remotas.
PSUICulture	Referencia cultural de la interfaz de usuario de la sesión de Windows PowerShell actual.
PSVersionTable	Información de versión de la sesión de PowerShell actual.
PWD	
ReportErrorShowExceptionClass	Especifica que los errores se muestren con una descripción de la clase de error.
ReportErrorShowInnerException	Especifica que se muestren los errores con las excepciones internas.
ReportErrorShowSource	Especifica que los errores se muestren con el origen del error.
ReportErrorShowStackTrace	Especifica que los errores se muestren con el seguimiento de la pila.
ShellId	El valor de ShellID identifica el shell actual. Lo usan las instrucciones #Requires.
StackTrace	
true	Boolean True
VerbosePreference	Indica la acción tomada cuando se entrega un mensaje detallado.
WarningPreference	Indica la acción tomada cuando se entrega un mensaje de advertencia.
WhatIfPreference	Si su valor es true, se considera WhatIf como habilitado para todos los comandos.

Para entender a súa función é moi interesante ler a súa descrición, pois as variables automáticas están moi ben documentadas.

1.6.1.7 Variables de entorno

As variables de entorno son unha fonte importante de información sobre o sistema operativo.

Para ler as variables de entorno hai que empregar **$env**: diante do nome da variable de entorno desexada. Por exemplo, se queremos ler a dirección da carpeta de Windows farémolo do seguinte xeito:

```
PS>$env:windir
C:\Windows
```

Se queres ver todas as variables de entorno existentes en Windows:

```
PS>DIR env:

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\Administrador\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   WIN-3E2I6RMDDEA
ComSpec                        C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK               NO
HOMEDRIVE                      C:
HOMEPATH                       \Users\Administrador
LOCALAPPDATA                   C:\Users\Administrador\AppData\Local
LOGONSERVER                    \\WIN-3E2I6RMDDEA
NUMBER_OF_PROCESSORS           1
OS                             Windows_NT
Path                           %SystemRoot%\system32\WindowsPowerShell\v1.0\;C:\Windows\system32;C:\Windows;C:\Windo...
PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE         AMD64
PROCESSOR_IDENTIFIER           Intel64 Family 6 Model 58 Stepping 9, GenuineIntel
PROCESSOR_LEVEL                6
PROCESSOR_REVISION             3a09
ProgramData                    C:\ProgramData
ProgramFiles                   C:\Program Files
ProgramFiles(x86)              C:\Program Files (x86)
ProgramW6432                   C:\Program Files
PSModulePath                   C:\Users\Administrador\Documents\WindowsPowerShell\Modules;C:\Windows\system32\Window...
PUBLIC                         C:\Users\Public
SESSIONNAME                    Console
SystemDrive                    C:
SystemRoot                     C:\Windows
TEMP                           C:\Users\ADMINI~1\AppData\Local\Temp
TMP                            C:\Users\ADMINI~1\AppData\Local\Temp
USERDOMAIN                     WIN-3E2I6RMDDEA
USERNAME                       Administrador
USERPROFILE                    C:\Users\Administrador
windir                         C:\Windows
windows_tracing_flags          3
windows_tracing_logfile        C:\BVTBin\Tests\installpackage\csilogfile.log
```

1.6.1.8 Ámbito das variables

As variables de PowerShell teñen un ámbito que determina onde unha variable é accesible. Así, os ámbitos soportados son:

- ***private*** - A variable créase só no ámbito actual e non pasa a outros ámbitos. Por conseguinte, só pode ser lida e escrita no ámbito actual.
- ***local*** - A variable créase só no ámbito local. Este é o modo por defecto empregado sen ámbito. Estas variables pódense ler dende ámbitos originarios do ámbito de aplicación actual, pero non se poden modificar.
- ***script*** - A variable será válida só no interior dun *script*, pero válida en calquera lugar del. Por exemplo, unha variable así definida no interior dunha función será accesible dende calquera outra función existente no interior dese *script*.
- ***global*** - A variable será válida en calquera lugar, incluso fora das funcións e *scripts*.

Estes ámbitos nos permiten restrinxir a visibilidade das variables en funcións ou *scripts*. Por defecto PowerShell fai unha **"restrición automática"** do ámbito das variables, é dicir, se unha variable foi creada no interior dun *script* non será accesible dende a liña de comandos. Así e todo, se queremos que as variables do interior dun *script* sexan accesibles dende a consola só temos que poñer un **.** diante da chamada do *script*:

```
#Fíxate que poñemos un "punto", un espazo e logo chamamos o script:
PS>. C:\scripts\test.ps1
#Outro xeito de facelo se estamos no directorio onde se atopa o script:
PS>. .\test.ps1
#Logo da execución do script as variables definidas nel son accesibles dende a consola.
```

Ollo! con este xeito de executar os *scripts* se este ten definidas "constantes".

Se queremos configurar ao noso antollo o ámbito das variables farémolo do seguinte xeito:

```
PS>$private:test1 = 100
PS>$local:test2 = 200
PS>$script:test3 = 300
PS>$global:test4 = 400
```

1.6.1.9 Tipos de varibles

PowerShell configura automáticamente o tipo de variable dependendo do dato gardado nela. Nun principio, nós non temos que facer nada. Saber que có comando **GetType().Name** podemos verificar o tipo de variable do que se trata. E tamén, tede en conta que podemos facer isto mesmo empregando "un valor" directamente:

```
#Directamente con DATOS:
PS> (12).GetType().Name
Int32
PS> (12.5).GetType().Name
Double
PS> ("H").GetType().Name
String
PS> (Get-Date).GetType().Name
DateTime
#Definir unha variable e logo ver o tipo que PS lle asigna:
PS> $numero = 100
PS> $numero.GetType().Name
Int32
```

Algúns dos tipos de variables cós que pode traballar PowerShell son os seguintes:

Tipo de Variable	Descrición	Exemplo
[array]	Un vector	
[bool]	Valor Si-Non	`[bool]$test = $true`
[byte]	Enteiro sen signo, 0...255	`[byte]$valor = 12`
[char]	Carácter Unicode individual	`[bool]$letra = "a"`
[datetime]	Data e Hora	`[datetime]$data = "24.Dec 2012 12:30"`
[decimal]	Número decimal	`[decimal]$num = 12`
[double]	Decimal punto flotante de doble precisión	`[double]$numero = 12.45`
[guid]	Número de identificación único de 32 bytes	
[hashtable]	Táboa hash	
[int16]	Enteiro con caracteres de 16 bits	`[int16]$test = 1000`
[int32] **[int]**	Enteiro con caracteres de 32 bits	`[int32]$test = 5000`
[int64] **[long]**	Enteiro con caracteres de 64 bits	`[int64]$test = 4GB`
[nullable]	Ensancha outro tipo de datos para engadirlles a capacidade de conter valores nulos. Pódese empregar, entre outros casos, para aplicar algún parámetro opcional	
[psobject]	Obxecto PowerShell	

Tipo de Variable	Descrición	Exemplo
[regex]	Expresión regular	`$texto = "Ola mundo"` `[regex]::split($texto, "la")`
[sbyte]	Enteiro con caracteres de 8 bits	`[sbyte]$valor = -12`
[scriptblock]	Bloque de código de PowerShell	
[single] **[float]**	Decimal punto flotante de simple precisión	`[single]$num = 44.76`
[string]	String	`[string]$texto = "Ola"`
[switch]	Parámetro de tipo "switch"	

[timespan]	Intervalo de tiempo	`PS> [timespan]$t = New-TimeSpan $(Get-Date)"1.Sep 07"`
		`PS> $t`
		`Days : -1944`
		`Hours : -14`
		`Minutes : -43`
		`Seconds : -17`
		`Milliseconds : -907`

[type]	Type	
[uint16]	Enteiro sen signo de 16 bits	`[uint16]$value = 1000`
[uint32]	Enteiro sen signo de 32 bits	`[uint32]$value = 1000`
[uint64]	Enteiro sen signo de 64 bits	`[uint64]$value = 1000`
[xml]	Documento XML	

PowerShell nos permite asignar o tipo de varible que imos gardar, farémolo poñendo diante da definición dela o tipo que nos interese gardar:

```
PS> [string]$test = 100
PS> $test
100
PS> $test.GetType().Name
String
```

Para convertir o tipo de varible temos que facer así:

```
PS> $num = 5000
PS> $num.GetType().Name
Int32
PS> $texto = [string]$num
PS> $texto
5000
PS> $texto.GetType().Name
String
```

Có seguinte exemplo vemos as vantaxes de empregar tipos de variables especializadas:

```
#Se non especificamos nada a seguinte variable será un string:
PS> $date.GetType().Name
String
PS> $date = "Diciembre 24, 2012"
PS> $date
Diciembre 24, 2012
PS> $date.AddDays(60)
Error en la invocación del método porque [System.String] no contiene ningún método llamado 'AddDays'.
En línea: 1 Carácter: 14
+ $date.AddDays <<<< (60)
    + CategoryInfo          : InvalidOperation: (AddDays:String) [], RuntimeException
    + FullyQualifiedErrorId : MethodNotFound
#Vemos que non nos permite empregar métodos específicos das variables de tipo "datetime"
```

```
#Borramos a variable e a volvemos a crear pero especificando que é do tipo desexado:
PS> del variable:\date
PS> [datetime]$date = "December 24, 2012"
PS C:\Users\Administrador> $date

lunes, 24 de diciembre de 2012 0:00:00


PS C:\Users\Administrador> $date.AddDays(60)

viernes, 22 de febrero de 2013 0:00:00
#Vemos que agora si podemos empregar os métodos específicos...
```

1.6.2 Arrays

Un *array* é unha variable que pode conter calquera número de elementos de calqueira tipo. O primeiro elemento ten a posición **0**, indicándose estes **separados por comas**. Vexamos un exemplo:

```
PS> $data = Get-Date
PS> $data

viernes, 28 de diciembre de 2012 20:02:34

PS> $numero = 25.40
PS> $numero
25,4
PS> $letra = "a"
PS> $letra
a
PS> $array = $data, "Ola mundo", $numero, $letra
PS> $array

viernes, 28 de diciembre de 2012 20:02:34
Ola mundo
25,4
a

PS>$array.GetType().Name
Object[]

#Outro xeito de crear un array:
PS>$array2 = @(1,2,3,"Hello")
#Crear un array baleiro:
PS>$array3 = @()
#Comprobar o tamaño actual do array:
PS> $array3.length
0
#Agregar un elemento ao array:
PS> $array3 += "Novo elemento"
PS> $array3.length
1
PS> $array3
Novo elemento
```

Como se comentou antes, os elementos dun *array* numéranse dende cero, e se pode acceder a calquera elemento indicando a súa posición entre corchetes **[]**.

```
PS> $array[2]
25,4
```

Para seleccionar varios elementos á vez:

```
PS> $array[0,2]
viernes, 28 de diciembre de 2012 20:02:34
25,4
```

Tamén se pode ver un rango de elementos indicando a posición de inicio e a de fin:

```
PS> $array[1..3]
Ola mundo
```

```
25,4
a
```

A posición do último elemento é **-1**:

```
PS> $array[-1]
a
```

E, tamén dicir, que podemos ver o número de elementos dun *array* có método **.count**.

```
PS> $array.count
4
```

A saída dun comando, aínda que nos pareza texto plano, é en realidade un *array*. Vexamos un exemplo:

```
#Gardamos o contido do comando ipconfig:
PS> $ip = ipconfig
#Comprobamos que se gardou:
PS> $ip

Configuración IP de Windows

Adaptador de Ethernet Conexión de área local:
   Sufijo DNS específico para la conexión. . : Home
   Vínculo: dirección IPv6 local. . . : fe80::84a9:ca57:4483:3184%11
   Dirección IPv4. . . . . . . . . . . . . . : 10.0.2.15
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 10.0.2.2
...
#Vemos se PowerShell creou un Array:
PS> $ip -is [Array]
True
#Calculamos o número de elementos do Array:
PS> $ip.Count
23
#Mostramos os elementos 7 e 8:
PS> $ip[7]
   Vínculo: dirección IPv6 local. . . : fe80::84a9:ca57:4483:3184%11
PS> $ip[8]
   Dirección IPv4. . . . . . . . . . . . . . : 10.0.2.15
```

Se algún comando se garda en texto plano e queremos forzar que se garde nun *array* empregaremos o constructor **@()**:

```
#Borramos a variable $ip antes creada
PS> del variable:\ip
#Forzamos a que a saída do comando ipconfig se garde nun array:
PS> $ip = @(ipconfig)
```

1.6.3 Listas

Unha lista é un *array* dinámico no que se poden engadir e eliminar elementos dun xeito moi fácil. Para definir unha lista baleira:

```
PS> $lista = New-Object System.Collections.ArrayList
```

Pódese ver que é un elemento **ArrayList** de .NET. Para engadir elementos de calquera tipo emprégase o método *.add*:

```
PS> $lista.add("cadea")
0
PS> $lista.add(45)
1
PS> $data = Get-date
PS> $lista.add($data)
2
PS> $lista.add("outra cadea")
3
PS> $lista.add(450)
4

#Para visualizar o seu contido:
```

```
PS> $lista
cadea
45
lunes, 31 de diciembre de 2012 4:37:51
outra cadea
450
```

Para eliminar elementos empregamos *.remove*, *.removerange* e *removeat*:

```
#Eliminar un elemento dado o seu contido:
PS> $lista.remove(45)
PS> $lista
cadea
lunes, 31 de diciembre de 2012 4:37:51
outra cadea
450

#Eliminar un rango de elementos:
PS> $lista.removerange(2,2)
PS> $lista
cadea
lunes, 31 de diciembre de 2012 4:37:51

#Eliminar un único elemento:
PS> $lista.removeat(0)
PS> $lista
lunes, 31 de diciembre de 2012 4:37:51
```

E, por último, indicar que, para eliminar todo o contido da lista e deixala baleira empregamos *.clear*:

```
PS> $lista.clear()
```

1.6.4 Táboas *hash*

As táboas *hash* gardan **"pares chave-valor"**. Para crear un *hash* baleiro:

```
PS> $hash = @{}
```

Pódense engadir pares chave-valor de calquera tipo:

```
PS> $hash["primeiro"] = 1
PS> $hash["data"] = $data
PS> $hash["taboa"] = 1,"dous",3,4.5
PS> $hash[2] = "dous"
```

Coma sempre, para ver o contido:

```
PS> $hash
Name                           Value
----                           -----
data                           31/12/2012 4:37:51
taboa                          {1, dous, 3, 4,5}
2                              dous
primeiro                       1
```

Para acceder ao valor dun elemento (chave), hai que indicar o nome do *hash* seguido dun punto e o nome da chave:

```
PS> $hash.primeiro
1
PS> $hash.data
lunes, 31 de diciembre de 2012 4:37:51
PS> $hash.taboa
1
dous
3
4,5
PS> $hash.2
Erro...
```

Vemos que aquí temos un erro pois a chave é un número. Isto soluciónase indicando a chave do elemento entre corchetes:

```
PS> $hash[2]
dous
```

Se as chaves son de tipo cadea tamén poden ir entre corchetes, pero, neste caso, a chave ten que ir entre comiñas:

```
PS> $hash["data"]
lunes, 31 de diciembre de 2012 4:37:51
```

É posible acceder só as chaves do *hash*:

```
PS> $hash.keys
data
taboa
2
primeiro
```

E tamén só aos valores:

```
PS> $hash.values
lunes, 31 de diciembre de 2012 4:37:51
1
dous
3
4,5
dous
1
```

Como un dos valores gardados é unha táboa, é posible acceder a cada un dos valores de dita táboa:

```
PS> $hash.taboa[0]
1
```

Para eliminar un dos elementos do *hash*:

```
PS> $hash.remove("data")
PS> $hash

Name                           Value
----                           -----
taboa                          {1, dous, 3, 4,5}
2                              dous
primeiro                       1
```

E para "limpar" todo o *hash*:

```
PS> $hash.clear()
PS> $hash
```

1.7 Obxectos

PowerShell traballa con obxectos, pero nos só vemos texto pois PowerShell "convirte" automáticamente eses obxectos en "texto".

Un obxecto non é máis que a suma de **Propiedades** e **Métodos**.

Se queremos ver as Propiedades e os Métodos dun obxecto empregaremos o *cmdlet* **Get-Member**. Vexamos como ver só as propiedades de, por exemplo, unha cadea de texto:

```
# Definimos a variable $cadea
PS> $cadea = "Ola caracola"
PS> $cadea
Ola caracola

# Listamos só as Propiedades:
PS> $cadea | Get-Member -memberType property
```

```
   TypeName: System.String

Name   MemberType Definition
----   ---------- ----------
Length Property   System.Int32 Length {get;}

# Listamos só os Métodos:
PS> $cadea | Get-Member -memberType method

   TypeName: System.String

Name             MemberType Definition
----             ---------- ----------
Clone            Method     System.Object Clone()
CompareTo        Method     int CompareTo(System.Object value), int CompareTo(string strB)
Contains         Method     bool Contains(string value)
CopyTo           Method     System.Void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count)
EndsWith         Method     bool EndsWith(string value), bool EndsWith(string value, System.StringComparison compari...
Equals           Method     bool Equals(System.Object obj), bool Equals(string value), bool Equals(string value, Sys...
GetEnumerator    Method     System.CharEnumerator GetEnumerator()
GetHashCode      Method     int GetHashCode()
GetType          Method     type GetType()
GetTypeCode      Method     System.TypeCode GetTypeCode()
IndexOf          Method     int IndexOf(char value), int IndexOf(char value, int startIndex), int IndexOf(char value...
IndexOfAny       Method     int IndexOfAny(char[] anyOf), int IndexOfAny(char[] anyOf, int startIndex), int IndexOfA...
Insert           Method     string Insert(int startIndex, string value)
IsNormalized     Method     bool IsNormalized(), bool IsNormalized(System.Text.NormalizationForm normalizationForm)
LastIndexOf      Method     int LastIndexOf(char value), int LastIndexOf(char value, int startIndex), int LastIndexO...
LastIndexOfAny   Method     int LastIndexOfAny(char[] anyOf), int LastIndexOfAny(char[] anyOf, int startIndex), int ...
Normalize        Method     string Normalize(), string Normalize(System.Text.NormalizationForm normalizationForm)
PadLeft          Method     string PadLeft(int totalWidth), string PadLeft(int totalWidth, char paddingChar)
PadRight         Method     string PadRight(int totalWidth), string PadRight(int totalWidth, char paddingChar)
Remove           Method     string Remove(int startIndex, int count), string Remove(int startIndex)
Replace          Method     string Replace(char oldChar, char newChar), string Replace(string oldValue, string newVa...
Split            Method     string[] Split(Params char[] separator), string[] Split(char[] separator, int count), st...
StartsWith       Method     bool StartsWith(string value), bool StartsWith(string value, System.StringComparison com...
Substring        Method     string Substring(int startIndex), string Substring(int startIndex, int length)
ToCharArray      Method     char[] ToCharArray(), char[] ToCharArray(int startIndex, int length)
ToLower          Method     string ToLower(), string ToLower(System.Globalization.CultureInfo culture)
ToLowerInvariant Method     string ToLowerInvariant()
ToString         Method     string ToString(), string ToString(System.IFormatProvider provider)
ToUpper          Method     string ToUpper(), string ToUpper(System.Globalization.CultureInfo culture)
ToUpperInvariant Method     string ToUpperInvariant()
Trim             Method     string Trim(Params char[] trimChars), string Trim()
TrimEnd          Method     string TrimEnd(Params char[] trimChars)
TrimStart        Method     string TrimStart(Params char[] trimChars)
```

1.8 Control de fluxo: condicións e bucles

1.8.1 Condicións

- **If**

Coma na maioría das linguaxes, if encárgase de avaliar e comparar casi todo. PowerShell emprega a estrutura básica: **if (condición) {acción}** e tamén admite **else** e **elseif**. Vexamos algún exemplo:

```
$linguaxe = "galego"

if ($linguaxe -eq "aleman")
{
"A linguaxe é alemán."
}
elseif ($linguaxe -eq "francés")
{
"A linguaxe é francés."
}
else
{
"A linguaxe non é nin alemán nin francés."
}
```

```
exit 0
```

Esta estrutura de avaliación de condicións sempre comeza con **if**, logo os **elseif** que se precisen e, por último, **else**, dependendo do camiño que queiras seguir. Neste exemplo, como **$linguaxe** non é nin "francés" sin "alemán", toma a acción de **else**.

Os operadores de comparación máis importantes:

Operador	Explicación
-eq	igual que
-gt	maior que
-ge	maior ou igual que
-lt	menor que
-le	menor ou igual que
-ne	distinto que
-like	parecido a (admite * como comodín)
-notlike	non parecido a (admite * como comodín)
-match	comprobar se existe unha cadea no interior de outra (admite expresións regulares)

Podemos ver un exemplo máis antes de seguir, nel compróbase a existencia dun arquivo compartido nun servidor e, se existe, este arquivo se executa coa aplicación de Windows predefinida para a súa extensión:

```
$servidor = "server00"
if (Test-path -path "\\$servidor\netlogon\aviso.html")
{
Invoke-Item "\\$servidor\netlogon\aviso.html"
}
```

Poden avaliarse dúas ou máis condicións cós operadores lóxicos **and** e **or**:

```
if  ( ( $linguaxe -ne "alemán" ) -and ( $linguaxe -ne "francés" ) )
```

Neste exemplo téñense que cumprir as dúas condicións ao redor de **and**, e hai que fixarse que todo debe ir dentro de paréntese.

- **Switch**

Se hai que avaliar moitas condicións á vez, é máis cómodo empregar **switch**. Vexamos un exemplo:

```
$linguaxe = "galega"

switch ($linguaxe)
{
"galega" {"A linguaxe é galega."}
"francés" {"A linguaxe é francés."}
"alemán" {"A linguaxe é alemán."}
"inglés" {"A linguaxe é inglés."}
"italiano" {"A linguaxe é italiano."}
"español" {"A linguaxe é español."}
"danés" {"A linguaxe é danés."}
default {"A linguaxe non está rexistrada."}
}

exit 0
```

- **Where**

Como vimos antes, podemos redirixir a saída dun *cmdlet*, obxecto ou colección de cousas por unha tubería e acceder as súas propiedades. Moitas veces necesitamos atopar algunha porpiedade concreta con algún valor concreto, e aquí é onde empreamos **where**. Vexamos un exemplo:

```
PS> Get-Date
miércoles, 02 de enero de 2013 16:15:25

PS> Get-Date | where {$_.year -eq 2012}
PS> Get-Date | where {$_.year -eq 2013}
miércoles, 02 de enero de 2013 16:16:01
```

Como vemos aquí, aparece a variable especial **$_**. Así que **Where** toma a saída do *cmdlet* e avalía o que aparece a continuación entre chaves. **$_** realmente é a saída de **get-date**, e **$_.year** é a propiedade ano de **get-date**. Despois, o operador **-eq** indica se **$_.year** é igual a 2012 na primeira e 2013 na segunda. Se se cumple a condición, devolve a saída en si como o faría **get-date**, e en caso contrario non devolve nada.

1.8.2 Bucles

PowerShell implementa 5 tipos de bucles: **foreach**, **for**, **while**, **do-while** e **do-until**. Funcionan con táboas, listas, *hashes* e calquera cousa que conteña unha colección de elementos.

- **Foreach**

É o máis sinxelo de usar, pois fai un percorrido directo por todos os elementos que lle des, por exemplo unha táboa:

```
$taboa = 1, 'dous', 3, 'catro', 5

foreach ($elemento in $taboa) {$elemento}

exit 0
```

Como pode verse, **foreach** crea un **$elemento** por cada elemento de **$taboa** e despois, entre chaves, podes facer calquera cousa con dito elemento, neste caso o único que se fai é mostralo.

- **For**

Crea e inicializa unha variable índice, avalía unha condición con esa variable e despois a incrementa se se cumple a condición.

```
PS> for ($i=1; $i -le 5; $i++) {$i}
1
2
3
4
5
```

- **While**

É igual que un **For** e tamén avalía a condición ao principio do bucle, pero non ten en conta nin a creación da variable índice nin o seu incremento.

```
PS> $i = 1
PS> while ($i -le 5) {$i; $i++}
1
2
3
4
5
```

Tamén pode ser a avaliación dunha cadea:

```
PS> $cadea = ""
PS> while ($cadea -ne "correcto") {$cadea = read-host "Contrasinal: "}
Contrasinal: : cousa
Contrasinal: : abc123.
Contrasinal: : correcto
PS>
```

- **Do-While**

Igual que **While** pero avalía a condición ao final de cada bucle. Vexamos un exemplo:

```
$cadea = ""
do
{
$cadea  = Read-Host "Contrasinal"
}
while ($cadea -ne "correcto")

exit 0
```

- **Do-Until**

Funciona exactamente igual que **Do-While** pero cambia a lóxica. **Do-While** da voltas mentres se produza a condición e **Do-Until** da voltas ata que se produza a condición. Vexamos un exemplo:

```
$cadea = ""
do
{
$cadea  = Read-Host "Contrasinal"
}
until ($cadea -eq "correcto")

exit 0
```

1.8.3 Bucles especiais

Pódese obter unha enumeración rápida do seguinte xeito:

```
PS> 1..5
1
2
3
4
5
```

Se redirixiramos esa saída por unha tubería, cada elemento da enumeración sae da tubería como a variable **$_** e podemos facer algo con iso, como por exemplo multiplicalo por 5:

```
PS> 1..5 | %{$_ * 5}
5
10
15
20
25
```

1.9 Funcións

Existen varias formas de definir funcións en PowerShell, vexamos a máis sinxela:

```
PS> function suma($x,$y) {$resultado = $x + $y; return $resultado}
PS> suma 2 3
5
PS>
```

Defínese a función suma que toma como parámetros as variables **$x** e **$y** (entre paréntese). A continuación, entre chaves fai algo con elas (neste caso as suma, coloca a solución en **$resultado** e devolve **$resultado**). A chamada á función é simple: escríbese o nome da función e, a continuación, indícanse os parámetros.

É interesante saber que tamén podemos crear unha variable có resultado da chamada á función:

```
PS> function suma($x,$y) { $x + $y }
PS> $resultado = suma 2 3
PS> $resultado
5
```

Podemos crear unha librería de funcións e chamala dende outros scripts.

Vexamos un exemplo:

◊ Creamos o arquivo "C:\scripts\funciones.ps1" có seguinte contido.

```
function sumar ($a, $b) {
    $a = [long]$a
    $b = [long]$b
    $resultado = $($a + $b)
    return $resultado
}

function multiplicar ($a, $b) {
    $a = [long]$a
    $b = [long]$b
    $resultado = $($a * $b)
    return $resultado
}

function restar ($a, $b) {
    $a = [long]$a
    $b = [long]$b
    $resultado = $($a - $b)
    return $resultado
}
```

◊ Creamos o arquivo "C:\scripts\proba.ps1" có seguinte contido.

```
#Chamamos o arquivo de funcións
. "C:\scripts\funciones.ps1"
#Pedimos que o usuario interaccione
$numa = Read-Host "Introduce primer número"
$numb = Read-Host "Introduce segundo número"
#Mostramos os resultados
Write-Host "$numa + $numb = $(sumar $numa $numb)"
Write-Host "$numa * $numb = $(multiplicar $numa $numb)"
Write-Host "$numa - $numb = $(restar $numa $numb)"
```

1.10 Arquivos

Neste punto veremos algunhas operacións con arquivos que se poden realizar con PowerShell.

Os *cmdlets* máis importantes para o manexo de arquivos son os seguintes:

Cmdlet	Descrición	Alias
Add-Content	Engade o contido a un arquivo.	**ac**
Clear-Host	Limpa a consola.	**cls, clear**
Clear-Item	Borra o contido do arquivo, pero non o arquivo.	**cli**
Copy-Item	Copia un arquivo ou un directorio.	**copy, cp, cpi**
Get-Childitem	Listar o contido de directorios.	**Dir, ls, gci**
Get-Content	Ler o contido de arquivos de texto.	**type, cat, gc**
Get-Item	Acceder a arquivos ou directorios.	**gi**
Get-ItemProperty	Ler as propiedades dun arquivo ou directorio.	**gp**
Invoke-Item	Abre un arquivo coa aplicación definida en Windows.	**ii**
Join-Path	Une dúas partes dunha ruta nunha única parte, por exemplo, un disco máis un nome de arquivo.	-

Cmdlet	Descrición	Alias
New-Item	Crea un novo arquivo ou directorio.	**ni**
Remove-Item	Borra moitos tipos de *items* como directorios, arquivos, variables, funcións, alias,...	**ri, rm, rmdir, del, erase, rd**
Rename-Item	Cambiarlle de nome a arquivos e directorios.	**ren, rni**
Resolve-Path	Resolve unha ruta incluídos os caracteres comodín.	**rvpa**
Set-ItemProperty	Establece a propiedade do arquivo ou directorio.	**sp**
Split-Path	Extraer unha parte específica dunha dirección.	-
Test-Path	Devolve *True* si existe unha dirección especificada.	-

1.10.1 Comprobación de arquivos

Se queremos comprobar a existencia dun arquivo, por exemplo, na carpeta actual:

```
PS> dir
    Directorio: C:\scripts

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        02/01/2013     23:37        100 proba.ps1

PS> Test-Path .\proba.ps1
True

PS> Test-Path .\arquivo.txt
False
```

1.10.2 Lectura rápida de arquivos de texto

O *cmdlet* **get-content** pode ler directamente o contido dun arquivo de texto especificado e pasa o seu contido a unha variable de tipo táboa. Cada liña lida do arquivo de texto será un elemento da táboa.

```
PS> $contido = Get-Content ./texto.txt
PS> $contido
primeira
segunda
terceira
PS> $contido[0]
primeira
PS> $contido[1]
segunda
PS> $contido[2]
terceira
PS> $contido[-1]
terceira
PS> $contido.count
3
```

1.10.3 Escritura rápida de arquivos de texto

Para escribir nun arquivo empregamos o *cmdlet* **out-file**.

Por exemplo, se queremos crear un arquivo de texto texto2.txt coa variable $contido antes empregada:

```
PS> $contido | Out-File ./texto2.txt
PS> cat .\texto2.txt
primeira
segunda
terceira
```

Se precisamos agregar unha cuarta liña no arquivo **texto2.txt**, farémolo así:

```
PS C:\scripts> "cuarta" | Out-File .\texto2.txt -append
PS C:\scripts> cat .\texto2.txt
primeira
segunda
terceira
cuarta
```

Con **Out-File** podemos crear/agregar o que nos interese, e se implica varias liñas, tamén:

```
PS> Get-Date | Format-List | Out-File .\texto2.txt -append
PS> cat .\texto2.txt
primeira
segunda
terceira
cuarta

DisplayHint : DateTime
Date        : 05/01/2013 0:00:00
Day         : 5
DayOfWeek   : Saturday
DayOfYear   : 5
Hour        : 23
Kind        : Local
Millisecond : 970
Minute      : 11
Month       : 1
Second      : 54
Ticks       : 634930243149708750
TimeOfDay   : 23:11:54.9708750
Year        : 2013
DateTime    : sábado, 05 de enero de 2013 23:11:54
```

1.10.4 Procura de cadeas en arquivos de texto

Entre as utilidades *cmdlets* exite unha denominada Select-String que nos permite buscar cadeas de texto dentro dunha variable ou un arquivo.

No seguinte exemplo buscamos a cadea te texto "segunda" no interior do arquivo "texto2.txt":

```
PS> Select-String "segunda" .\texto2.txt

texto2.txt:2:segunda
```

Como vemos, o resultado da procura indica o nome do arquivo, o número de liña onde atopou o que se buscaba, e a cadea que se estaba a buscar.

Tamén se pode buscar

```
PS> Get-ChildItem -filter *.txt -recurse | Select-String "segunda"

texto.txt:2:segunda
texto2.txt:2:segunda
```

1.11 Expresións regulares

Como xa sabemos de Linux, as expresións regulares son o xeito máis potente e efectivo de atopar patróns de caracteres en cadeas de texto. PowerShell utiliza o operador -match para este propósito.

```
PS > $cadea = "Ola caracola"
PS > $cadea
Ola caracola
PS > $cadea -match "cara"
True
PS > $cadea -match "peixe"
False

# Podemos empregar as expresións regulares para comprobar se o patrón existe ou non:
PS > $cadea -match "cara.o"
```

```
True
PS > $cadea -match "cara[a-z]+"
True
```

- **Distinguir entre maiúsculas e minúsculas:** Hai que ter en conta que **match** non distingue entre maiúsculas e minúsculas nas súas procuras. Se precisamos facer esta distinción, empregaremos o operador **cmatch**.

```
PS > $cadea -match "cara[A-Z]+"
True
PS > $cadea -cmatch "cara[A-Z]+"
False
```

- **Extraer patróns:** Cando **match** ou **-cmatch** atopa un patrón nunha variable dada, PowerShell crea un hash novo chamado **$matches** para gardar o atopado:

```
PS > $cadea -match "cara[A-Z]+"
True
PS > $matches
Name                           Value
----                           -----
0                              caracola

PS > $matches[0]
caracola
```

- **O problema das expresións regulares tendo máis dunha coincidencia:** Se empregando o operador **-match** con expresións regulares temos máis dunha coincidencia este operador NON traballará ben.

```
PS > $ip = ipconfig
PS > $ip

Configuración IP de Windows

Adaptador de Ethernet Conexión de área local:

   Sufijo DNS específico para la conexión. . : Home
   Vínculo: dirección IPv6 local. . . : fe80::486a:b9c9:bee:fff0%11
   Dirección IPv4. . . . . . . . . . . . . . : 10.0.2.15
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 10.0.2.2

Adaptador de túnel isatap.Home:

   Estado de los medios. . . . . . . . . . . : medios desconectados
   Sufijo DNS específico para la conexión. . : Home

Adaptador de túnel Teredo Tunneling Pseudo-Interface:

   Sufijo DNS específico para la conexión. . :
   Dirección IPv6 . . . . . . . . . . . . . . : 2001:0:5ef5:73b8:2c7a:4eb:f5ff:fdf0
   Vínculo: dirección IPv6 local. . . : fe80::2c7a:4eb:f5ff:fdf0%14
   Puerta de enlace predeterminada . . . . . : ::
PS > $ip -match '([0-9]{1,3}\.){3}[0-9]{1,3}'
   Dirección IPv4. . . . . . . . . . . . . . : 10.0.2.15
   Máscara de subred . . . . . . . . . . . . : 255.255.255.0
   Puerta de enlace predeterminada . . . . . : 10.0.2.2
PS > $matches
PS >
```

Para solucionar isto temos que empregar o obxecto RegEx do seguinte xeito:

```
PS > $patronip = [regex] "([0-9]{1,3}\.){3}[0-9]{1,3}"

PS > $patronip.Matches($ip)
Groups   : {10.0.2.15, 2.}
Success  : True
Captures : {10.0.2.15}
Index    : 245
Length   : 9
Value    : 10.0.2.15
```

```
Groups   : {255.255.255.0, 255.}
Success  : True
Captures : {255.255.255.0}
Index    : 302
Length   : 13
Value    : 255.255.255.0

Groups   : {10.0.2.2, 2.}
Success  : True
Captures : {10.0.2.2}
Index    : 363
Length   : 8
Value    : 10.0.2.2

PS > $patronip.Matches($ip) | Select-Object -Property Value
Value
-----
10.0.2.15
255.255.255.0
10.0.2.2

PS > $patronip.Matches($ip) | ForEach-Object {"IP: $($_.Value)" }
IP: 10.0.2.15
IP: 255.255.255.0
IP: 10.0.2.2

PS >
```

Con respecto á distinción entre maiúsculas e minúsculas dicir que, o obxecto **RegEx** as distingue por defecto, polo que, para solucionalo debemos poñer "(?i)" diante da expresión regular. Por exemplo un patrón para un mail sería:

```
PS > $patronmail = [regex] "(?i)\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b"
PS >
```

Visitar este enlace interesante.

- **Substitucións:**

O operador -replace é o empregado en PowerShell para remprazar cadeas de texto.

```
PS > "Ola Patricia" -replace "Ola", "Adeus"
Adeus Patricia
```

Tamén podemos empregar expresións regulares e, neste caso, cada un dos resultados obtidos serán gardados por PowerShell en variables temporais chamadas $1, $2, $3...

```
PS C:\Users\Administrador> "Ola Patricia" -match '(.*) (.*)'
True
PS C:\Users\Administrador> $matches

Name                           Value
----                           -----
2                              Patricia
1                              Ola
0                              Ola Patricia

PS > "Ola Patricia" -replace '(.*) (.*)', '$2 $1'
Patricia Ola
PS >
```

1.12 Administración do sistema

O punto forte de PowerShell é a administración do sistema operativo.

1.12.1 Obter información do sistema

Para obter información do sistema podemos empregar, entre outras as seguintes variables e cmdlets:

- **$env:computername** ---- Mostra o nome do equipo.
- **$env:username** ---- Mostra o nome do usuario logeado nese intre.
- **$env:userdomain** ---- Mostra o dominio no que se iniciou sesión.
- **Get-Date** ---- Mostra a data e hora actual do equipo.
- **Get-HotFix** ---- Mostra información sobre actualizacións e Service Packs do equipo.
- **Get-WmiObject -Class Win32_ComputerSystem** ---- Mostrar información do equipo.
    - Exemplos.
- **Get-WmiObject -Class Win32_bios** ---- Mostrar información da BIOS do equipo.

```
#Buscar os equipos que teñen BIOS distinta ao equipo A05
$comp = get-content computers.txt
$comp | foreach {
$bios = get-wmiobject -class win32_bios -computername $_
$BiosVersion = $bios.SMBIOSBIOSVersion
if (!($BiosVersion ?match ?A05?)) {
add-content (?$_ $BiosVersion?) -path checkfailed.txt }
}
}
```

- **Get-WmiObject -Class Win32_PhysicalMemory** ---- Mostrar información da Memoria instalada no equipo ou equipos.
- **Get-WmiObject -Class Win32_Processor** ---- Mostrar información do Procesador existente no equipo ou equipos.
- **Get-WmiObject -Class Win32_DiskDrive** ---- Mostrar información do Discos Duros existentes no equipo ou equipos.
    - ◊ **Get-WmiObject -Class Win32_DiskPartition** ---- Mostrar información das Particións existente no equipo ou equipos.
    - ◊ **Get-WmiObject -Class Win32_logicaldisk** ---- Mostrar información das Unidades lóxicas (C: D: E: ...) existente no equipo ou equipos.
- **Get-WmiObject -Class Win32_OperatingSystem** ---- Mostrar información do sistema operativo.
- **Get-WmiObject -Class Win32_systemdriver** ---- Mostrar información dos drivers instalados no sistema.

```
#Se queremos ver só os drivers que se atopan "correndo"
PS> get-wmiobject -class win32_systemdriver -filter ?state=?Running??
```

- **Get-WmiObject -Class Win32_UserAccount** ---- Mostrar información das contas de usuario existentes no equipo ou no dominio ao que pertence o equipo.
- **Get-WmiObject -Class Win32_Group** ---- Mostrar información dos grupos existentes no equipo ou no dominio ao que pertence o equipo.

1.12.2 Os Proveedores

Os comandos de manipulación de arquivos nos permiten tamén xestionar:

- O rexistro.
- As variables.
- Alias.
- A base de datos de certificados X509.
- Funcións.
- O sistema de arquivos.

Así temos oito "proveedores" (*Provider* ou *PSProvider*) e, para conocelos, podemos teclear o comando ***Get-PsProvider***.

```
PS> Get-PSProvider
Name                Capabilities                                Drives
----                ------------                                ------
Alias               ShouldProcess                               {Alias}
Environment         ShouldProcess                               {Env}
FileSystem          Filter, ShouldProcess, Credentials          {C, D}
Function            ShouldProcess                               {Function}
Registry            ShouldProcess, Transactions                 {HKLM, HKCU}
```

```
Variable            ShouldProcess                                   {Variable}
```

Así, por exemplo, ver as variables de contorno:

```
PS C:\Users\usuario> Get-ChildItem Env:

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\usuario\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   PORTATIL
ComSpec                        C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK               NO
HOMEDRIVE                      C:
HOMEPATH                       \Users\usuario
LOCALAPPDATA                   C:\Users\usuario\AppData\Local
LOGONSERVER                    \\PORTATIL
MOZ_PLUGIN_PATH                C:\Program Files\Tracker Software\PDF Viewer\Win32\
NUMBER_OF_PROCESSORS           4
OS                             Windows_NT
Path                           C:\ProgramData\Oracle\Java\javapath;C:\Program Files (x86)\Intel\iCLS Client\;C:\Prog...
PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL
PROCESSOR_ARCHITECTURE         AMD64
PROCESSOR_IDENTIFIER           Intel64 Family 6 Model 69 Stepping 1, GenuineIntel
PROCESSOR_LEVEL                6
PROCESSOR_REVISION             4501
ProgramData                    C:\ProgramData
ProgramFiles                   C:\Program Files
ProgramFiles(x86)              C:\Program Files (x86)
ProgramW6432                   C:\Program Files
PSModulePath                   C:\Users\usuario\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShe...
PUBLIC                         C:\Users\Public
SESSIONNAME                    Console
SystemDrive                    C:
SystemRoot                     C:\Windows
TEMP                           C:\Users\usuario\AppData\Local\Temp
TMP                            C:\Users\usuario\AppData\Local\Temp
USERDOMAIN                     portatil
USERDOMAIN_ROAMINGPROFILE      portatil
USERNAME                       usuario
USERPROFILE                    C:\Users\usuario
VBOX_MSI_INSTALL_PATH          C:\Program Files\Oracle\VirtualBox\
windir                         C:\Windows
```

Podemos tamén crear unha variable nova e logo eliminala:

```
# Creamos a variable varTest có contido "Variable de Test"
PS C:\Users\usuario> Set-Location Env:
PS Env:\> New-Item -Path . -Name varTest -Value 'Variable de test'

Name                           Value
----                           -----
varTest                        Variable de test

# Buscamos todas as variables de Env que teñan no seu nome "Test"
PS Env:\> Get-ChildItem env: | where-Object {$_.Name -match "Test"} | fl
Name  : varTest
Value : Variable de test

# Comprobamos que existe a variable varTest
PS Env:\> Test-Path Env:varTest
True

# Recuperamos o seu contido (valor)
PS Env:\> $Env:varTest
Variable de test

# Eliminamos a variable varTest
PS Env:\> Remove-Item Env:\varTest
```

```
# Comprobamos que xa non existe
PS Env:\> Test-Path Env:varTest
False
```

1.12.3 Apagar e reiniciar o equipo

Podemos empregar os seguintes cmdlets para apagar e reiniciar o equipo:

```
# Apagar o equipo
PS> Stop-Computer

# Reiniciar o equipo
PS> Restart-Computer
```

1.12.4 Instalar aplicacións e Actualizar o sistema

Para realizar estas tarefas empregaremos o Administrador de paquetes Chocolatey.

```
# Recuerda activar la ejecución de scripts ps1 en el equipo:
PS>Set-ExecutionPolicy RemoteSigned
# Instalamos Chocolatey:
PS>iex ((new-object net.webclient).DownloadString('https://chocolatey.org/install.ps1'))
# Podemos probar xa a instalar algún programa, como o Firefox:
PS>choco install firefox
# Para actualizalo:
PS>choco upgrade firefox
# Instalamos o módulo Windows Update PowerShell, que nos permitirá actualizar o Sistema:
PS>choco install pswindowsupdate
# Podemos ver a lista de actualizacións:
PS> Get-WUList -WindowsUpdate
# E tamén descargalas e actualizalas:
PS>Get-WUInstall -WindowsUpdate -AcceptAll -AutoReboot
```

Enlace interesante.

Módulos Powershell para manexo de paquetes:

- Technet Web

```
# Instalar paquete
PS> Find-Package -Name virtualbox -Provider chocolatey | Install-Package
# Desinstalar paquete
PS>Uninstall-Package -name "openoffice"
```

Tamén podemos instalar aplicacións, actualizalas ou executar calquera comando REMOTAMENTE. Vexamos como facelo:

- Executar comandos remotos en Windows

A execución remota de comandos emprega o protocolo WS-Management.
Os requisitos dos equipos locales e remotos para executar comandos remotos podemos velos aquí.

- Windows PowerShell 2.0 o posterior.
- Microsoft .NET Framework 2.0 o posterior.
- Versión 2.0 de Administración remota de Windows.

Así, no equipo remoto:

- Activar o *script* remoto:

```
PS> Enable-PSRemoting
```

- Para ver si está activado:

```
PS> New-PSSession
 Id Name            ComputerName    State         ConfigurationName     Availability
 -- ----            ------------    -----         -----------------     ------------
  1 Session1        localhost       Opened        Microsoft.PowerShell     Available
```

- No equipo local, dende onde nos conectaremos, activar a execución de scripts:

```
PS> Set-ExecutionPolicy RemoteSigned
```

- E, xa por último, executar dende este equipo o script no equipo remoto empregando o comando Invoke-Command:

```
PS>  Invoke-Command -ComputerName wclient1.viso.local -filepath C:\scripts\instalacion.ps1
```

Un exemplo de *script* pode ser o seguinte, que instala chocolatey e logo o Firefox, o Google Chrome e o VLC:

```
set-executionpolicy RemoteSigned -Confirm:$false
iex ((new-object net.webclient).DownloadString('https://chocolatey.org/install.ps1'))
choco install firefox -y
choco install googlechrome -y
choco install vlc -y
```

Configuración do mirror de paquetes de Chocolatey para evitar baneos en instalacións masivas nas aulas dos paquetes de Chocolatey:

Neste caso imos facer uso do servidor mirror.sanclemente.local do instituto. O comando a empregar é o seguinte:

```
choco source add -n=nexus -s=http://mirror.sanclemente.local:8081/repository/chocolatey-group/
```

A partir deste momento a descarga de paquetes farase empregando o mirror. Non será necesario polo tanto configurar o proxy para que Chocolatey se conecte a Internet.

Configuración dun proxy de rede para que Chocolatey se conecte a Internet:

Enlace

```
# Por exemplo:
PS>choco config set proxy http://10.0.0.1:3128
```

1.12.5 Nome do equipo

Tarefas sinxelas como descubrir ou cambiar o nome do equipo fanse do seguinte xeito empregando o acceso a obxectos de **WMI (Instrumental de Administración de Windows)**.

- Para descubrir o nome dun equipo:

```
# Descubrir o nombre dun equipo
## PowerShell 1.0 e 2.0
#PS > $equipo = Get-WmiObject win32_computersystem
## PowerShell 3.0 e posteriores
PS > $equipo = Get-CimInstance -ClassName Win32_ComputerSystem
PS > $equipo | fl

Domain              : iessantiago.local
Manufacturer        : innotek GmbH
Model               : VirtualBox
Name                : WSERVER1
PrimaryOwnerName    : Usuario de Windows
TotalPhysicalMemory : 1073270784

PS > $equipo.name
WSERVER1
```

- Para cambiar o nome dun equipo no PowerShell 1.0 e 2.0:

```
# Cambiar o nome dun equipo no PowerShell 1.0 e 2.0
# Creamos o obxecto $nombreequipo
PS > $equipo = Get-WmiObject win32_computersystem

# Vemos cal é o nome actual:
PS > $equipo.name
WIN-3E2I6RMDDEA

# Pedimos ao usuario o novo nome do equipo
```

```
# e o gardamos na variable $name:
PS > $name = Read-Host -Prompt "Introduce nome para o equipo"
Introduce nome para o equipo: Server00
PS > $name
Server00

# Configuramos o novo nome có método Rename do obxecto ComputerSystem:
PS > $equipo.Rename($name)

__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0

# Para que o cambio funcione hai que reiniciar o equipo:
PS > Restart-Computer
```

- Para cambiar o nome dun equipo no PowerShell 3.0 e posterior:

```
## PowerShell 3.0 e posteriores
PS > Rename-Computer -NewName WCLIENT1 -Restart -Force
```

1.12.6 Configuración IP

- Para ver cal é a configuración de rede do teu equipo podemos facelo do seguinte xeito:

```
# Crear o obxecto que representa a tarxeta de rede:
## PowerShell 1.0 e 2.0
#PS > $adaptador = Get-WmiObject Win32_NetworkAdapterConfiguration | Where-Object { $_.IpEnabled }
## PowerShell 3.0 e posteriores
PS > $adaptador = Get-CimInstance -ClassName Win32_NetworkAdapterConfiguration | Where-Object { $_.IpEnabled }
PS > $adaptador

DHCPEnabled      : False
IPAddress        : {192.168.1.100, fe80::84a9:ca57:4483:3184}
DefaultIPGateway : {192.168.1.1}
DNSDomain        :
ServiceName      : E1G60
Description      : Adaptador de escritorio Intel(R) PRO/1000 MT
Index            : 7

# Recorda:
# - Ver todas as propiedades do obxecto:
#### $adaptador | get-member -membertype property
# - Ver todos os métodos do obxecto:
#### $adaptador | get-member -membertype method

PS C:\> $adaptador.IPAddress
192.168.1.100
fe80::84a9:ca57:4483:3184

# Conseguir a IP:
PS C:\> $adaptador.IPAddress[0]
192.168.1.100

# Conseguir a MAC:
PS C:\> $adaptador.MACAddress
08:00:27:EC:98:04
```

- Para cambiar a configuración IP do equipo facémolo do seguinte xeito...

    ◊ PowerShell 1.0 e 2.0:

```
# Parámetros a configurar:
```

```
$IP = "10.1.100.0"
$MS = "255.255.0.0"
$PE = "10.1.0.254"
$DNS1 = "10.4.0.1"
$DNS2 = "10.4.0.2"

# Crear o boxecto que representa a tarxeta de rede:
$adaptador = Get-WmiObject Win32_NetworkAdapterConfiguration | Where-Object { $_.IpEnabled }

# Cambiarlle a configuración IP ao equipo
# ------------------------------------------------------------ #
# Así configuramos unha IP estática:
$adaptador.EnableStatic($IP,$MS)
$adaptador.SetGateways($PE, [UInt16] 1)
$arrDNSServers = @($DNS1, $DNS2)
$adaptador.SetDNSServerSearchOrder($arrDNSServers)

# ------------------------------------------------------------ #
# Se queres configurar a IP por DHCP:
$adaptador.EnableDHCP()
$arrDNSServers = @()
$adaptador.SetDNSServerSearchOrder($arrDNSServers)
```

◊ PowerShell 3.0 e posteriores (cmdlets para a administración da rede):

```
#Eliximos o adaptador que imos configurar
#Get-NetAdapter
#Unha vez que o temos claro...
$nic = "Ethernet"
$IP = "172.16.10.10"
$MS = 24
$PE = "172.16.10.1"
$DNS1 = "10.0.4.1"
$DNS2 = "10.0.4.2"
$arrDNSServers = @($DNS1, $DNS2) #Temos que facer un array cós servidores DNS

# Desabilitamos o DHCP se fose necesario
Set-NetIPInterface -InterfaceAlias $nic -DHCP Disabled

# Eliminamos a configuración IP do equipo se existe
Remove-NetIPAddress -InterfaceAlias $nic -IncludeAllCompartments -Confirm:$false  2> $null
# Eliminamos a configuración da Porta de Enlace se existe
Remove-NetRoute -InterfaceAlias $nic -Confirm:$false  2> $null

# Configuramos a IP estática, a máscara de subrede e a porta de enlace
New-NetIPAddress -InterfaceAlias $nic -AddressFamily IPv4 -IPAddress $IP -PrefixLength $MS -Type Unicast -DefaultGateway $PE

# Configuramos os servidores DNS
Set-DnsClientServerAddress -InterfaceAlias $nic -ServerAddresses $arrDNSServers
```

1.12.7 Procesos e Servizos

- **Procesos.** A memoria de Windows, como calquera outro sistema operativo, serve para aloxar procesos que corren no sistema. Có *cmdlet* Get-Process de PowerShell podemos ver todos os procesos que actualmente están correndo no sistema:

```
PS > Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     43       5     1952       4980    53     0,22   1940 conhost
    277      10     1688       3724    42     0,38    320 csrss
    175      11     1876       5316    43     0,67    360 csrss
     66       7     1304       4188    49     0,02   1876 dwm
    489      33    13088      25544   159     0,41   1900 explorer
      0       0        0         24     0               0 Idle
    532      19     3448       9456    38     0,17    464 lsass
    143       7     2136       3800    18     0,03    472 lsm
    143      17     3216       7276    60     0,05   1584 msdtc
    554      23    48884      48340   563     1,05   1892 powershell
    184      12     3728       7584    30     0,47    456 services
     29       2      364       1032     5     0,09    244 smss
```

```
    261      18     6064      10260    73     0,08   1032 spoolsv
    154       8     5092      11520    36     2,23   1640 sppsvc
    287      32     6980      11192    46     0,17     12 svchost
    340      13     3220       8012    41     0,13    564 svchost
     68       6     1352       4300    29     0,03    632 svchost
    214      15     2672       6248    30     0,05    688 svchost
    289      15     7984      10800    43     0,20    776 svchost
    816      37    15428      27356   390     2,13    812 svchost
    284      22     5788      12040    59     0,17    856 svchost
    192      14     3280       9028    58     0,05    908 svchost
    398      26    10440      14328   335     0,20    948 svchost
     46       4      780       2620    13     0,00   1076 svchost
    461       0      112        300     3               4 System
    134      11     2700       5944    51     0,02   1492 taskhost
    110       9     1808       5008    35     0,02    624 VBoxService
     82       8     1392       4240    67     0,06   2000 VBoxTray
     77       9     1316       4176    43     0,13    368 wininit
     96       7     1496       4760    27     0,06    396 winlogon

PS >
```

Se queremos ver a información relevante do proceso **powershell**, indicamos o seu nome:

```
PS > Get-Process powershell
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    537      23    55300      55356   563     1,09   1892 powershell

PS >
```

Unha das propiedades que nos da *Get-Process* do proceso é **WorkingSet** que é o tamaño en bytes do proceso en cuestión:

```
PS > (Get-Process powershell).ws
57688064
PS > (Get-Process powershell).ws /1mb
55,01953125
PS >
```

Outra propiedade interesante é a ruta onde está o arquivo do proceso en cuestión. Por exemplo, se tempos o Block de notas aberto e queremos ver onde está o seu arquivo en disco:

```
PS > (Get-Process notepad).path
C:\Windows\system32\notepad.exe
PS >
```

Para mostrar varias propiedades dun proceso, empregamos o *cmdlet* Format-Table có seu argumento *property* e indicamos todas as propiedades que nos interesen separadas por comas:

```
PS > Get-Process powershell | Format-Table name, company -auto
Name       Company
----       -------
powershell Microsoft Corporation

PS >
```

O argumento *-auto* axusta o ancho das columnas ao ancho dos datos visualizados. Se queremos visualizar moitas propiedades tamén podemos empregar Format-List.

Tamén podemos ver todos os procesos que teñan unha propiedade común. Por exemplo, se queremos ver todos os procesos pertencentes á compañía "Microsoft":

```
PS > Get-Process | Where-Object {$_.company -like '*Microsoft*'}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     43       5     1952       4980    53     0,27   1940 conhost
    263      10     1688       3724    42     0,38    320 csrss
     66       7     1304       4188    49     0,02   1876 dwm
    491      33    13184      25740   159     0,44   1900 explorer
```

```
    527      19     3448       9456    38     0,17    464 lsass
    140       7     2136       3800    18     0,03    472 lsm
    143      17     3216       7276    60     0,05   1584 msdtc
     72       7     1420       5016    71     0,03   1400 notepad
    528      23    49692      49872   563     1,22   1892 powershell
    180      12     3672       7564    29     0,47    456 services
     29       2      364       1032     5     0,09    244 smss
    261      18     6012      10244    73     0,08   1032 spoolsv
    152       8     5040      11488    35     2,28   1640 sppsvc
    291      32     6984      11196    46     0,17     12 svchost
    135      11     2700       5960    51     0,02   1492 taskhost
     77       9     1316       4176    43     0,13    368 wininit
     96       7     1496       4760    27     0,06    396 winlogon

PS >
```

Tamén podemos ver unha lista dos procesos agrupados e ordenados pola compañía á que pertencen:

```
PS > Get-Process | Group-Object company | Sort-Object name | Format-List

Name   :
Count  : 2
Group  : {System.Diagnostics.Process (Idle), System.Diagnostics.Process (System)}
Values : {$null}

Name   : Microsoft Corporation
Count  : 27
Group  : {System.Diagnostics.Process (conhost), System.Diagnostics.Process (csrss), System.Diagnostics.Process (csrss),
          System.Diagnostics.Process (dwm)...}
Values : {Microsoft Corporation}

Name   : Oracle Corporation
Count  : 2
Group  : {System.Diagnostics.Process (VBoxService), System.Diagnostics.Process (VBoxTray)}
Values : {Oracle Corporation}

PS >
```

Por último, podemos finalizar un proceso de dous xeitos:

```
#Modo 1:
PS > Get-Process notepad | Stop-Process

#Modo 2:
PS > (Get-Process notepad).kill()

PS >
```

- **Servizos.** Un servizo é un proceso que se inicia no sistema antes de que o usuario inicie a súa sesión. Son típicos os servizos de Microsoft como a cola de impresión, o cliente DHCP, o cliente DNS, compartición de carpetas, audio, actualizacións de Windows, conexións de rede, etc. Outros fabricantes tamén aportan servizos ao sistema cando os seus produtos son instalados: antivirus, controladores, servidores, etc.

Podemos ver os servizos iniciados có *cmdlet* Get-Service.

```
PS > Get-Service

Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Experiencia con aplicaciones
Stopped  ALG                Servicio de puerta de enlace de niv...
Stopped  AppIDSvc           Identidad de aplicación
Stopped  Appinfo            Información de la aplicación
Stopped  AppMgmt            Administración de aplicaciones
Stopped  AudioEndpointBu... Compilador de extremo de audio de W...
Stopped  AudioSrv           Audio de Windows
Running  BFE                Motor de filtrado de base
Running  BITS               Servicio de transferencia inteligen...
Stopped  Browser            Examinador de equipos
Stopped  CertPropSvc        Propagación de certificados
Stopped  clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
Stopped  clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
```

```
Stopped  COMSysApp          Aplicación del sistema COM+
Running  CryptSvc           Servicios de cifrado
Running  DcomLaunch         Iniciador de procesos de servidor DCOM
Stopped  defragsvc          Desfragmentador de disco
Running  Dhcp               Cliente DHCP
Stopped  Dnscache           Cliente DNS
Stopped  dot3svc            Configuración automática de redes c...
...
```

Este comando nos devolve o estado, o nome e a descrición de cada un dos servizos.

Como vemos, o estado do servizo pode ser **Running** (en execución) ou **Stopped** (parado). Os servizos parados normalmente dependen de outros servizos para iniciarse. Para ver os servizos e as súas dependencias empregamos o argumento **ServicesDependedOn**.

```
PS > Get-Service | Format-Table -property name, servicesdependedon -auto

Name                           ServicesDependedOn
----                           ------------------
AeLookupSvc                    {}
ALG                            {}
AppIDSvc                       {CryptSvc, RpcSs, AppID}
Appinfo                        {RpcSs, ProfSvc}
AppMgmt                        {}
AudioEndpointBuilder           {PlugPlay}
AudioSrv                       {AudioEndpointBuilder, RpcSs, MMCSS}
BFE                            {RpcSs}
BITS                           {EventSystem, RpcSs}
Browser                        {LanmanServer, LanmanWorkstation}
CertPropSvc                    {RpcSs}
clr_optimization_v2.0.50727_32 {}
clr_optimization_v2.0.50727_64 {}
COMSysApp                      {EventSystem, SENS, RpcSs}
CryptSvc                       {RpcSs}
DcomLaunch                     {}
defragsvc                      {RPCSS}
Dhcp                           {Afd, Tdx, NSI}
Dnscache                       {nsi, Tdx}
dot3svc                        {Ndisuio, RpcSs, Eaphost}
DPS                            {}
EapHost                        {RPCSS, KeyIso}
EFS                            {RPCSS}
eventlog                       {}
EventSystem                    {rpcss}
...
```

E para ver os servizos que dependen de cada un empregaríamos **DependentServices**:

```
PS > Get-Service | Format-Table -property name, dependentservices -auto

Name                           DependentServices
----                           -----------------
AeLookupSvc                    {}
ALG                            {}
AppIDSvc                       {}
Appinfo                        {}
AppMgmt                        {}
AudioEndpointBuilder           {AudioSrv}
AudioSrv                       {}
BFE                            {SharedAccess, RemoteAccess, PolicyAgent, MpsSvc...}
BITS                           {}
Browser                        {}
CertPropSvc                    {}
clr_optimization_v2.0.50727_32 {}
clr_optimization_v2.0.50727_64 {}
COMSysApp                      {}
CryptSvc                       {AppIDSvc}
DcomLaunch                     {wuauserv, WPDBusEnum, WinRM, SharedAccess...}
defragsvc                      {}
Dhcp                           {WinHttpAutoProxySvc}
Dnscache                       {}
dot3svc                        {}
DPS                            {}
```

```
EapHost                         {dot3svc}
EFS                             {}
...
```

Podemos parar un servizo en calquera momento con **Stop-Service**:

```
PS > Stop-Service spooler
```

E tamén inicialo con **Start-Service**:

```
PS > Start-Service spooler
```

1.12.8 O sistema de arquivos

1.12.8.1 Discos Duros: Particionar, Instalar FS e Configurar Quotas

Para administrar discos temos unha serie de cmdlets interesantes, ainda que, a día de hoxe, case todas as accións as podemos seguir realizando có domando Diskpart.

Podemos ver un exemplo onde se vai a particionar un disco (**disk 1** neste caso)do seguinte xeito:
- 1 Partición Primaria de 10GB, formateada en NTFS, montada na letra P: e con etiqueta "DatosNTFS".
- 1 Partición Extendida que ocupe o resto do espazo libre do disco duro.
    - 1 Partición Lóxica de 5GB, formateada en FAT32, montada na letra L: e con etiqueta "DatosFAT".

```
# Creamos o contido do arquivo "discos.txt" onde estarán os comandos de diskpart.
# O contido deste arquivo será o seguinte:
SELECT DISK 1
CLEAN
CREATE PARTITION PRIMARY SIZE=10240
ASSIGN LETTER=P
FORMAT FS=NTFS LABEL="DatosNTFS" QUICK
CREATE PARTITION EXTENDED
CREATE PARTITION LOGICAL SIZE=5120
ASSIGN LETTER=L
FORMAT FS=FAT32 LABEL="DatosFAT" QUICK
####
#Dende a liña de comandos PowerShell chamamos ao comando diskpart:
PS>diskpart /s ./discos.txt
```

Vexamos algún comando interesante de Powershell:

- ***get-psdrive*** : Mostra os discos PowerShell que están dispoñibles.

```
PS> Get-PSDrive | fl Name
Name : Alias
Name : C
Name : Cert
Name : D
Name : Env
Name : Function
Name : HKCU
Name : HKLM
Name : V
Name : Variable
Name : W
Name : WSMan
```

- ***Initialize-Disk*** :
- ***Clear-Disk*** :
- ***New-Partition*** :
- ***format-volume*** : Un cmdlet interesante que nos permite formatear un volume xa existente:

```
PS> Format-Volume -DriveLetter W -FileSystem NTFS -Force -Confirm:$false
```

Exemplo de particionamento dun disco crear nel dúas particións primarias (60% do tamaño e 40% do tamaño a outra).

◊ A primeira en NTFS e a segunda en exFAT
◊ Montadas en V: e W:

```
$numero = 2 #Temos que saber o disco duro que imos particionar
$disco = Get-Disk -Number $numero
$tamdiscob = $disco.Size
$tampartV = [long]$tamdiscob * 60 / 100 # La partición V: será el 60% del disco
                                        # La W: el resto que quede
if ($disco.PartitionStyle -eq "RAW") {
    Initialize-Disk -Number $numero -PartitionStyle MBR
}
Clear-Disk -Number $numero -RemoveData -Confirm:$false
Initialize-Disk -Number $numero -PartitionStyle MBR
New-Partition -DiskNumber $numero -DriveLetter V -Size $tampartV
New-Partition -DiskNumber $numero -DriveLetter W -UseMaximumSize

Format-Volume -DriveLetter V -FileSystem NTFS -NewFileSystemLabel "Datos NTFS" -Force -Confirm:$false
Format-Volume -DriveLetter W -FileSystem exFAT -NewFileSystemLabel "Datos exFAT" -Force -Confirm:$false
```

- **Para configurar as Quotas de disco** farémolo có comando Fsutil que realiza tarefas relacionadas có sistema de arquivos NTFS.

  - Para obter unha lista das Quotas de disco existentes para un volumen ou partición:

  ```
  PS> Fsutil quota query D:
  ```

  - Para configurar Quota no disco D: e facer que os membros do grupo G-Triton00, pertencente ao Dominio Triton00, teñan un nivel de advertencia de 25MB e un límite de 30MB:

  ```
  PS> Fsutil quota enforce D:
  PS> Fsutil quota modify D: 26214400 31457280 Triton00\G-Triton00
  ```

- Enlace interesante de Administración de discos con Powershell.
- Cmdlets para administrar o almacenamento en discos.

1.12.8.2 Arquivos e carpetas

Pódese dicir que, os arquivos son contenedores básicos de información e as carpetas son grupos de arquivos e carpetas.

- Para ver a ruta onde estamos actualmente na estrutura de carpetas empregamos a variable **$pwd** ou, simplemente **pwd**:

  ```
  PS > $pwd
  Path
  ----
  C:\scripts

  PS > $pwd | Get-Member

     TypeName: System.Management.Automation.PathInfo

  Name         MemberType Definition
  ----         ---------- ----------
  Equals       Method     bool Equals(System.Object obj)
  GetHashCode  Method     int GetHashCode()
  GetType      Method     type GetType()
  ToString     Method     string ToString()
  Drive        Property   System.Management.Automation.PSDriveInfo Drive {get;}
  Path         Property   string Path {get;}
  Provider     Property   System.Management.Automation.ProviderInfo Provider {get;}
  ProviderPath Property   string ProviderPath {get;}

  PS >
  ```

  A ruta é a propiedade **path**:

  ```
  PS > $pwd.path
  C:\scripts

  PS >
  ```

Conseguimos o mesmo resultado có *cmdlet* Get-Location.

Para cambiar a posición na árbore de ficheiros empregamos o *cmdlet* Set-Location.

- Podemos obter un listado de todos os arquivos e carpetas dun directorio en particular empregando o *cmdlet* Get-Childitem.

```
PS > Get-Childitem -force

    Directorio: C:\

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d--hs        05/10/2011     15:06            $Recycle.Bin
d--hs        05/10/2011     15:05            Archivos de programa
d--hs        14/07/2009      7:06            Documents and Settings
d----        14/07/2009      5:20            PerfLogs
d-r--        05/10/2011     15:09            Program Files
d-r--        14/07/2009      7:06            Program Files (x86)
d--h-        05/10/2011     15:05            ProgramData
d--hs        05/10/2011     15:05            Recovery
d----        17/12/2012     20:08            scripts
d--hs        05/10/2011     14:58            System Volume Information
d-r--        05/10/2011     15:05            Users
d----        06/01/2013     12:20            Windows
-a-hs        06/01/2013     13:14 1073741824 pagefile.sys

PS >
```

O argumento **-force** visualiza todos os arquivos, incluidos os arquivos ocultos e de sistema. Como dato interesante, aparecen na columna da esquerda os 5 modos que poden ter cada arquivo mostrado. Estes modos son:

d (*directory*): carpeta
a (*archive*): arquivo
r (*read only*): só lectura
h (*hidden*): oculto
s (*system*): arquivo de sistema

Para obter toda a estrutura de arquivos e carpetas dun directorio dado empregamos o argumento **-recurse**.

```
PS > Get-ChildItem -force -recurse

    Directorio: C:\Users\Administrador\Documents

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d--hs        05/10/2011     15:05            Mi música
d--hs        05/10/2011     15:05            Mis imágenes
d--hs        05/10/2011     15:05            Mis vídeos
-a-hs        17/12/2012     19:23        402 desktop.ini
...
```

Tamén podemos empregar comodíns utilizando o argumento **-filter**.

```
PS > Get-ChildItem -filter *.PDF -recurse
```

- Podemos buscar cadeas de texto dentro de arquivos empregando o *cmdlet* Select-String.

```
PS C:\> Get-ChildItem -recurse | where { $_ | Select-String "ola" -quiet }
```

- ***New-Item*** : Para crear arquivos e carpetas.

```
#Crear o directorio "Datos" en D:\
PS> new-item -type directory -path D:\Datos
#Para crear o directorio "Datos" en D:\ dos equipos EQ01, EQ02 e EQ03
PS> invoke-command -computername EQ01, EQ02, EQ03 -scriptblock { new-item -type directory -path D:\Datos }
```

- ***Copy-Item*** : Para copiar arquivos e carpetas.

```
#Copiar un arquivo:
```

```
PS> copy-item arquivo.txt arquivo-copia.txt

#Copiar un directorio:
PS> copy-item D:\Datos Z:\Datos-backup -recurse

#Copiar só os arquivos pdf:
PS> copy-item D:\Datos\*.pdf Z:\Datos-backup -recurse
```

Sempre e cando se teñan os permisos axeitados, deberíase deser capaz de mover directorios e arquivos.
Pero hai que ter en conta o seguinte: debido a que non puede utilizar ***Move-Item*** para mover recursos a través de volumes, a ruta de orixe e destino debe ter raíces idénticas. Se os arquivos dun directorio están en uso, un archivo está en uso ou se comparte un directorio, no se poderá mover o directorio ou o arquivo.

1.12.8.3 ACLs en volumes NTFS

Nas unidades NTFS, os "permisos de acceso" determinan qué usuarios poden acceder aos arquivos e directorios. Para cada arquivo e directorio, os datos de seguridade forman o denominado **"descritor de seguridade" (SD)**. Os permisos de acceso reais están nas **"listas de control de acceso (ACL)"**. As ACLs divídense nun conxunto de **"entradas de control de acceso (ACEs)"**.

Introdución aos permisos NTFS.

Comentar por último, antes de comezar, que temos así dous tipos de permisos:

- DACL: Parte do descritor de seguridade dun obxecto que concede ou denega a usuarios e grupos específicos o permiso de acceso ao obxecto.
- SACL: Parte do descriptor de seguridade dun obxexto que especifica os eventos que se auditarán por usuario ou grupo. Algúns exemplos de eventos de auditoría son o acceso a arquivos, os intentos de inicio de sesión e os peches do sistema.

Para descubrir a configuración dos permisos NTFS empregaremos o *cmdlet* Get-Acl.

```
# Creamos a carpeta datos
PS > md datos
# Chequeamos a configuración de seguridade do directorio
PS > Get-Acl .\datos | fl

Path   : Microsoft.PowerShell.Core\FileSystem::C:\datos
Owner  : BUILTIN\Administradores
Group  : SERVIDOR\None
Access : NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Administradores Allow  FullControl
         BUILTIN\Usuarios Allow  ReadAndExecute, Synchronize
         BUILTIN\Usuarios Allow  AppendData
         BUILTIN\Usuarios Allow  CreateFiles
         CREATOR OWNER Allow  268435456
Audit  :
Sddl   : O:BAG:S-1-5-21-763290472-4206933357-2196511326-513D:AI(A;OICIID;FA;;;SY)(A;OICIID;FA;;;BA)(A;OICIID;0x1200a9;;
         ;BU)(A;CIID;LC;;;BU)(A;CIID;DC;;;BU)(A;OICIIOID;GA;;;CO)
```

Para ver os permisos de arquivos e sudirectorios do interior dun directorio de xeito recursivo, podemos empregar algo parecido a isto:

```
PS C:\> Get-ChildItem c:\datos -recurse | get-acl | foreach { $_.path; $_.Access | ft -wrap }
Microsoft.PowerShell.Core\FileSystem::C:\datos\cosas

   FileSystemRights   AccessControlType IdentityReference           IsInherited    InheritanceFlags    PropagationFlags
   ----------------   ----------------- -----------------           -----------    ----------------    ----------------
        FullControl               Allow NT AUTHORITY\SYSTEM                True  ContainerInherit,                None
                                                                                     ObjectInherit
        FullControl               Allow BUILTIN\Administrad                True  ContainerInherit,                None
                                        ores                                         ObjectInherit
   ReadAndExecute,                Allow BUILTIN\Usuarios                   True  ContainerInherit,                None
        Synchronize                                                              ObjectInherit
         AppendData                Allow BUILTIN\Usuarios                   True    ContainerInherit               None
        CreateFiles                Allow BUILTIN\Usuarios                   True    ContainerInherit               None
          268435456                Allow CREATOR OWNER                      True  ContainerInherit,         InheritOnly
                                                                                     ObjectInherit


Microsoft.PowerShell.Core\FileSystem::C:\datos\archivo.txt
```

```
   FileSystemRights   AccessControlType IdentityReference          IsInherited    InheritanceFlags    PropagationFlags
   ----------------   ----------------- -----------------          -----------    ----------------    ----------------
        FullControl              Allow NT AUTHORITY\SYSTEM               True              None                None
        FullControl              Allow BUILTIN\Administrad               True              None                None
                                       ores
   ReadAndExecute,               Allow BUILTIN\Usuarios                  True              None                None
        Synchronize


Microsoft.PowerShell.Core\FileSystem::C:\datos\cosas\archivo2.txt

   FileSystemRights   AccessControlType IdentityReference          IsInherited    InheritanceFlags    PropagationFlags
   ----------------   ----------------- -----------------          -----------    ----------------    ----------------
        FullControl              Allow NT AUTHORITY\SYSTEM               True              None                None
        FullControl              Allow BUILTIN\Administrad               True              None                None
                                       ores
   ReadAndExecute,               Allow BUILTIN\Usuarios                  True              None                None
        Synchronize
```

O mellor nestes casos é gardar a saída do *cmdlet* **Get-Acl** nunha variable (que será en realidade un obxecto **System.Security.AccessControl** que representa as ACLs). Unha vez creado o obxecto podemos ler a propiedade ou manexar o método que nos interese:

```
PS C:\> $permisos = Get-Acl C:\datos
PS C:\> $permisos.Access | ft -wrap

   FileSystemRights   AccessControlType IdentityReference          IsInherited    InheritanceFlags    PropagationFlags
   ----------------   ----------------- -----------------          -----------    ----------------    ----------------
        FullControl              Allow NT AUTHORITY\SYSTEM               True  ContainerInherit,                None
                                                                                    ObjectInherit
        FullControl              Allow BUILTIN\Administrad               True  ContainerInherit,                None
                                       ores                                         ObjectInherit
   ReadAndExecute,               Allow BUILTIN\Usuarios                  True  ContainerInherit,                None
        Synchronize                                                                 ObjectInherit
         AppendData              Allow BUILTIN\Usuarios                  True    ContainerInherit              None
        CreateFiles              Allow BUILTIN\Usuarios                  True    ContainerInherit              None
          268435456              Allow CREATOR OWNER                     True  ContainerInherit,         InheritOnly
                                                                                    ObjectInherit
```

- Unha vez creado o obxecto, é moi interesante a opción que nos da o PowerShell de gardalo nun arquivo XML para o seu almacenamento. Logo poderemos empregar este arquivo XML para "clonar" as ACLs noutro directorio ou arquivo.

```
PS> $acl = Get-Acl c:\datos
PS> $acl | Export-Clixml c:\permisos.xml
```

Para configurar as ACLs dende un obxecto temos o *cmdlet* Set-Acl. Empregando o obxecto creado no exemplo anterior con **Get-Acl**:

```
# Primeiro creamos o obxecto:
PS> $acl = Get-Acl c:\datos
# Agora clonamos o obxecto ACL nos dun novo directorio:
PS> Set-Acl -Path C:\datos2 $acl
```

Tamén podemos facelo, tal e como vimos antes, cun paso intermedio empregando un arquivo XML:

```
# Gardamos o obxecto ACL directamente nun arquivo:
PS > Get-Acl c:\datos | Export-Clixml c:\permisos.xml
# Ese arquivo o podemos gardar, cambiar de equipo, etc
# Importamos o arquivo nun novo obxecto ACL:
PS > $acl = Import-Clixml C:\permisos.xml
# Aplicamos ese obxecto a un novo directorio ou arquivo:
PS > Set-Acl -Path C:\datos2 $acl
```

Para configurar os permisos dende PowerShell temos os comandos **CACLS**, **XCACLS** e, moito mellor, ICACLS. Veremos, a continuación, o seu funcionamento:

- Para ver a axuda do comando:

```
PS> icacls /?
ICACLS nombre /save archivoACL [/T] [/C] [/L] [/Q]
```

```
    almacena las DACL para los archivos y carpetas cuyos nombres coinciden
    en archivoACL para su uso posterior con /restore. Tenga en cuenta que no
    se guardan las SACL, el propietario ni las etiquetas de identidad.

ICACLS directorio [/substitute SidOld SidNew [...]] /restore archivoACL
                  [/C] [/L] [/Q]
    aplica las DACL almacenadas a los archivos del directorio.
...
```

- ICACLS permítenos almacenar nun arquivo as DACLs para os arquivos e carpetas que nos interesen empregando o parámetro /**save**, logo poderemos recuperar os permisos empregando o parámetro /**restore**:

```
# ICACLS nome /save arquivoACL [/T] [/C] [/L] [/Q]
PS C:\> icacls C:\datos\* /save permisos.txt /T
archivo procesado: C:\datos\archivo.txt
archivo procesado: C:\datos\cosas
archivo procesado: C:\datos\cosas\archivo2.txt
Se procesaron correctamente 3 archivos; error al procesar 0 archivos
PS C:\> Get-Content permisos.txt
a r c h i v o . t x t

 D : A I ( A ; I D ; F A ; ; ; S Y ) ( A ; I D ; F A ; ; ; B A ) ( A ; I D ; 0 x 1 2 0 0 a 9 ; ; ; B U )

 c o s a s

 D : A I ( A ; O I C I I D ; F A ; ; ; S Y ) ( A ; O I C I I D ; F A ; ; ; B A ) ( A ; O I C I I D ; 0 x 1 2 0 0 a 9 ;
; ; B U ) ( A ; C I I D ; L C ; ; ; B U ) ( A ; C I I D ; D C ; ; ; B U ) ( A ; O I C I I O I D ; G A ; ; ; C O )

 c o s a s \ a r c h i v o 2 . t x t

 D : A I ( A ; I D ; F A ; ; ; S Y ) ( A ; I D ; F A ; ; ; B A ) ( A ; I D ; 0 x 1 2 0 0 a 9 ; ; ; B U )


PS C:\>
```

- Vexamos a recuperación dos permisos antes gardados no arquivo, empregamos para isto o comando /**restore**:

```
# ICACLS directorio [/substitute SidOld SidNew [...]] /restore archivoACL [/C] [/L] [/Q]
# Para restaurar todas as ACL gardadas no arquivo permisos.txt
# en todos os subdirectorios e arquivos que exista en C:\datos
PS C:\> icacls C:\datos\ /restore ./permisos.txt
archivo procesado: C:\datos\archivo.txt
archivo procesado: C:\datos\cosas
archivo procesado: C:\datos\cosas\archivo2.txt
Se procesaron correctamente 3 archivos; error al procesar 0 archivos
PS C:\>

# Fixádevos que agora non se pon o * despois de C:\datos\
```

- Vamos a configurar a carpeta de exemplo *C:\datos* cunha serie de permisos:

- Administrador: Control Total na carpeta, subcarpetas e arquivos.
- Usuarios: Só Crear carpetas nesa carpeta.
- Cando crea un usuario na carpeta o usuario terá control total sobre ela. Ningún outro usuario poderá entrar nesa carpeta.

```
#Primeiro miramos os permisos iniciais:
PS C:\> get-acl c:\datos | foreach { $_.path; $_.Access | ft -wrap }
Microsoft.PowerShell.Core\FileSystem::C:\datos

   FileSystemRights   AccessControlType IdentityReference           IsInherited    InheritanceFlags    PropagationFlags
   ----------------   ----------------- -----------------           -----------    ----------------    ----------------
        FullControl              Allow NT AUTHORITY\SYSTEM                True  ContainerInherit,                None
                                                                               ObjectInherit
        FullControl              Allow BUILTIN\Administrad                True  ContainerInherit,                None
                                       ores                                    ObjectInherit
   ReadAndExecute,               Allow BUILTIN\Usuarios                   True  ContainerInherit,                None
        Synchronize                                                            ObjectInherit
         AppendData              Allow BUILTIN\Usuarios                   True    ContainerInherit               None
        CreateFiles              Allow BUILTIN\Usuarios                   True    ContainerInherit               None
          268435456              Allow CREATOR OWNER                      True  ContainerInherit,          InheritOnly
                                                                               ObjectInherit
```

```
# Agora configuramos os permisos...
# Na seguinte liña rompemos a herdanza, borrando ademais todos os permisos anteriores,
# o Administrador terá permiso de control total,
# ese permiso se propagará logo a arquivos e carpetas existentes no interior dese directorio C:\datos
PS C:\> icacls C:\datos /inheritance:r /grant:r "Administrador:`(OI`)`(CI`)F"
archivo procesado: C:\datos
Se procesaron correctamente 1 archivos; error al procesar 0 archivos
# Hai que fixarse que no PowerShell, antes dos parénteses, hai que colocar a comiña esa... senón, non vai.

#Vexamos como van os pemisos ata o de agora:
PS C:\> Get-Acl C:\datos | ft -wrap

    Directorio: C:\

Path                                Owner                               Access
----                                -----                               ------
datos                               BUILTIN\Administradores             SERVIDOR\Administrador Allow
                                                                        FullControl

# Agora engadimos o "Creator Owner" con Control Total.
PS C:\> icacls C:\datos /grant:r "CREATOR OWNER:`(OI`)`(CI`)F"
archivo procesado: C:\datos
Se procesaron correctamente 1 archivos; error al procesar 0 archivos

# Quédanos engadir o grupo usuarios con permisos de crear só carpetas nesta carpeta.
PS C:\> icacls c:\datos /grant:r "usuarios:`(NP`)`(AD,RD`)"
archivo procesado: c:\datos
Se procesaron correctamente 1 archivos; error al procesar 0 archivos
PS C:\> get-acl c:\datos | fl

Path   : Microsoft.PowerShell.Core\FileSystem::C:\datos
Owner  : BUILTIN\Administradores
Group  : SERVIDOR\None
Access : CREATOR OWNER Allow  FullControl
         BUILTIN\Administradores Allow  FullControl
         BUILTIN\Usuarios Allow  ReadData, AppendData
         SERVIDOR\Administrador Allow  FullControl

# Vemos que aparece, sen saber por que, un permiso ao grupo Administradores que non debería estar ahí,
# vamos a eliminalo:
PS C:\> icacls C:\datos /remove:g "Administradores"
archivo procesado: C:\datos
Se procesaron correctamente 1 archivos; error al procesar 0 archivos
PS C:\> get-acl c:\datos | fl


Path   : Microsoft.PowerShell.Core\FileSystem::C:\datos
Owner  : BUILTIN\Administradores
Group  : SERVIDOR\None
Access : CREATOR OWNER Allow  FullControl
         BUILTIN\Usuarios Allow  ReadData, AppendData
         SERVIDOR\Administrador Allow  FullControl
```

1.12.9 Crear recursos compartidos e conectarse a eles

Para compartir carpetas na rede emprégase o comando net share. Con PowerShell 3.0 e posteriores podemos empregar tamén os cmdlets do módulo SMB Share como veremos logo:

```
PS> net share scripts=d:\scripts /grant:"Todos,FULL" /CACHE:None
# Para ver os recursos compartidos:
PS> net share
```

Para conectarse a un recurso compartido dende outro equipo empregamos o comando net use.

```
PS> net use S: \\server\scripts /persistent:no
```

- Para PowerShell 3.0 e posteriores empregaremos os *cmdlets* do módulo SMB Share e podes obter moita axuda no seguinte enlace:

```
# Ver a lista dos cmdlets do múdulo
PS> Get-Command -module smb*
```

```
CommandType     Name                                               ModuleName
-----------     ----                                               ----------
Alias           Move-SmbClient                                     SmbWitness
Function        Block-SmbShareAccess                               SmbShare
Function        Close-SmbOpenFile                                  SmbShare
Function        Close-SmbSession                                   SmbShare
Function        Disable-SmbDelegation                              SmbShare
Function        Enable-SmbDelegation                               SmbShare
Function        Get-SmbBandwidthLimit                              SmbShare
Function        Get-SmbClientConfiguration                         SmbShare
Function        Get-SmbClientNetworkInterface                      SmbShare
Function        Get-SmbConnection                                  SmbShare
Function        Get-SmbDelegation                                  SmbShare
Function        Get-SmbMapping                                     SmbShare
Function        Get-SmbMultichannelConnection                      SmbShare
Function        Get-SmbMultichannelConstraint                      SmbShare
Function        Get-SmbOpenFile                                    SmbShare
Function        Get-SmbServerConfiguration                         SmbShare
Function        Get-SmbServerNetworkInterface                      SmbShare
Function        Get-SmbSession                                     SmbShare
Function        Get-SmbShare                                       SmbShare
Function        Get-SmbShareAccess                                 SmbShare
Function        Get-SmbWitnessClient                               SmbWitness
Function        Grant-SmbShareAccess                               SmbShare
Function        Move-SmbWitnessClient                              SmbWitness
Function        New-SmbMapping                                     SmbShare
Function        New-SmbMultichannelConstraint                      SmbShare
Function        New-SmbShare                                       SmbShare
Function        Remove-SmbBandwidthLimit                           SmbShare
Function        Remove-SmbMapping                                  SmbShare
Function        Remove-SmbMultichannelConstraint                   SmbShare
Function        Remove-SmbShare                                    SmbShare
Function        Revoke-SmbShareAccess                              SmbShare
Function        Set-SmbBandwidthLimit                              SmbShare
Function        Set-SmbClientConfiguration                         SmbShare
Function        Set-SmbPathAcl                                     SmbShare
Function        Set-SmbServerConfiguration                         SmbShare
Function        Set-SmbShare                                       SmbShare
Function        Unblock-SmbShareAccess                             SmbShare
Function        Update-SmbMultichannelConnection                   SmbShare

# Ver os recursos compartidos existentes no equipo:
PS> Get-SmbShare | fl

Name        : ADMIN$
ScopeName   : *
Path        : C:\Windows
Description : Admin remota

Name        : C$
ScopeName   : *
Path        : C:\
Description : Recurso predeterminado

Name        : IPC$
ScopeName   : *
Path        :
Description : IPC remota

# Creamos un novo directorio e logo o compartimos para Todos - Control Total:
PS> mkdir C:\COSA
PS>  New-SmbShare -Name COSA -Path C:\COSA -FullAccess Todos -Description "Cousas para compartir"
PS> Get-SmbShare | fl
...
Name        : COSA
ScopeName   : *
Path        : C:\COSA
Description : Cousas para compartir
...

# Para eliminar o recurso compartido:
PS> Remove-SmbShare -Name COSA -Force
```

Para conectarse a un recurso compartido dende outro equipo, empregando PowerShell 3.0 ou posterior, empregamos os seguintes *cmdlets*:

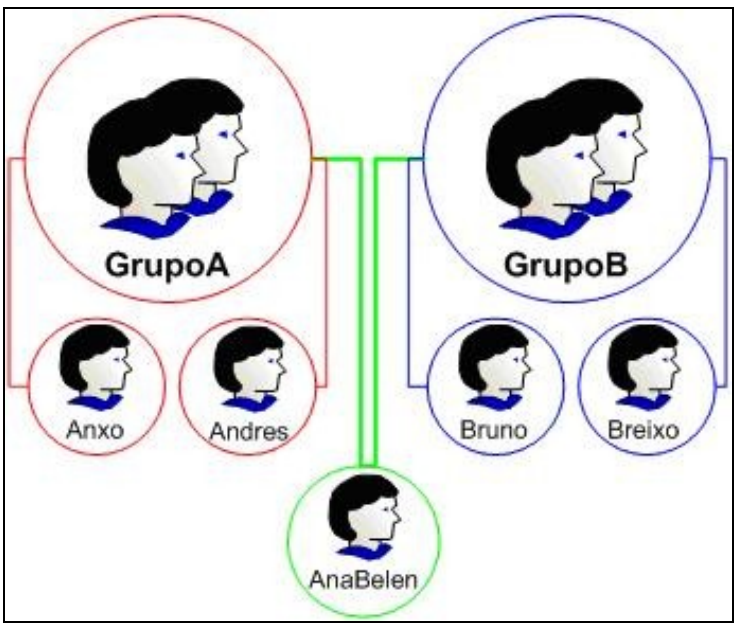
```
# Para conectarse:
PS> New-SmbMapping -LocalPath S: -RemotePath \\W12Base\COSA$ -Persistent $False
Status                              Local Path                          Remote Path
------                              ----------                          -----------
OK                                  S:                                  \\W12Base\COSA$

# Para ver os recursos compartidos existentes:
PS> Get-SmbMapping
Status                              Local Path                          Remote Path
------                              ----------                          -----------
OK                                  S:                                  \\W12Base\COSA$

# Para eliminar un recurso compartido existente:
PS> Remove-SmbMapping -LocalPath S: -Force
```

1.12.10 Usuarios y Grupos del equipo local

Resumen de los cmdlets existentes para administrar los Usuarios y Grupos del equipo local:

- Add-LocalGroupmember: agrega un miembro en un grupo local.
- Disable-LocalUser: desactivar cuenta de usuario.
- Enable-LocalUser: activar cuenta de usuario.
- Get-LocalGroup: mostrar conjunto de grupos de seguridad locales presentes en el puesto de trabajo.
- Get-LocalGroupMember: recuperar los miembros presentes en un grupo local.
- Get-LocalUser: mostrar el conjunto de cuentas de usuario locales presentes en el puesto de trabajo.
- New-LocalGroup: crear un nuevo grupo de seguridad.
- New-LocalUser: crear una nueva cuenta de usuario local.
- Remove-LocalGroup: elimina un grupo de seguridad local.
- Remove-LocalGroupMember: elimina un miembro de un grupo local
- Remove-LocalUser: elimina una cuenta de usuario local.
- Rename-LocalGroup: renombra un grupo de seguridad local.
- Rename-LocalUser: renombra una cuenta de usuario local.
- Set-LocalGroup: permite modificar las propiedades de un grupo de seguridad local.
- Set-LocalUser: permite modificar las propiedades de una cuenta de usuario local.

Y podemos ver un ejemplo donde se dan de alta los siguientes Usuarios y Grupos Locales:




```
#Tabla hash con lista de Grupos y Usuarios
$GruposUsuarios = @{"GrupoA"=@("Anxo","Andrea","AnaBelen"); "GrupoB"=@("Bruno","Breixo","AnaBelen")}
#Preparamos la contraseña ya cifrada
$password = "abc123.."
$pwd = ConvertTo-SecureString $password -AsPlainText -Force
```

```
#Iteramos en los grupos a dar de alta
foreach ($grupo in $GruposUsuarios.keys) {
    #Damos de alta el grupo
    New-LocalGroup -Name $grupo -Description "Miembros del $grupo"

    #Iteramos en los miembros de cada grupo
    foreach ($usuario in $GruposUsuarios[$grupo]) {
        #Hacemos un try - catch para no ver el error de dar altasde usuario repetidas
        try {
            #Agregamos el usuario al equipo
            New-LocalUser -Name $usuario -Password $pwd 2> $null
        }
        catch {
            "El usuario $usuario ya existe"
        }
        #Añadimos el usuario al grupo correspondiente
        Add-LocalGroupMember -Group $grupo -Member $usuario
    }
}
```

1.13 A Seguridade

Está claro que hai que ter moito coidado cós scripts, pois a facilidade de facer cousas pode levarnos a unha situación peligrosa para o noso sistema.

1.13.1 A seguridade de PowerShell por defecto

PowerShell trae unhas normas de seguridade por defecto que son as seguintes:

- Os arquivos ps1 están asociados ao bloc de notas. Podemos asocialos ao PowerShell, pero hai que ter moito coidado, pois é moi sinxelo facer un doble clic e executar un script que, en realidade, queríamos editar.
- Unha política de execución limitada, tal e como vimos no primeiro punto desta axuda (Por defecto está en *Restricted* e o pasamos a *RemoteSigned*).

1.14 Active Directory

Os Servizos de Dominio de Active Directory (AD DS) proporcionan a funcionalidade dunha solución de Identidade e Acceso para redes de empresas. Esta infraestructura realizará o seguinte:

- Almacenar información sobre usuarios, grupos, ordenadores e outras identidades.
- Autenticar unha identidade.
- Control de acceso.
- Proporcionar un rastreo de auditoría.

O *Active Directory* emprega:

- LDAP (Lightweight Directory Access Protocol).
- Kerberos
- DNS

É interesante comprender o significado dos termos **DN (*Distinguished Name*)**, **RDN (*Relative Distinguished Name*)** y **CN (*Common Name*)**:

- Os **DN** son un tipo de ruta cara un obxecto en *Active Directory*. Cada obxecto en AD ten un único DN. Un usuario chamado Brais Piñeiro, que está no interior da OU Direccion, que, á súa vez, está no interior da OU Triton00 e pertence ao dominio Triton00.local, ten o DN:

    CN=Brais Piñeiro,OU=Direccion,OU=Triton00,DC=Triton00,DC=local.

  - O DN é un camiño a partires do obxecto ata o máis alto nivel do dominio no nome de espazo DNS triton00.local.
  - O CN trátase do nome común do usuario, é dicir, o seu nome completo.
  - OU é a Unidade Organizativa.
  - E, por último, DC significa Compoñente de Dominio.

- O fragmento do DN anterior ao primeiro OU ou contedor chámase **"Nome distintivo Relativo" (RDN)**. No caso de Brais, o RDN do obxecto é CN=Brais.

- Como o DN dun obxecto debe ser único dentro do servizo de directorio, o RDN dun obxecto debe ser único dentro do seu contedor.

1.14.1 Instalación dos Servizos de Dominio de Active Directory

- Para instalar estes servizos, é dicir, promocionar o equipo Windows 2008 a Servidor de Dominio empregaremos a aplicación dcpromo.exe.

- No Windows 2012 a aplicación *dcpromo* non existe, o xeito de promocionar o equipo a Servidor de Dominio será o seguinte:

```
# Instalar os servizos de Active Directory no servidor:
PS > Install-WindowsFeature ?Name AD-Domain-Services ?ComputerName NomedoEquipo
# Promocionar o servidor:
PS > Invoke-Command ?ComputerName NomedoEquipo
?ScriptBlock {Import-Module ADDSDeployment;Install-ADDSForest
?CreateDNSDelegation:$False ?DatabasePath ?C:\Windows\NTDS? ?DomainMode ?Win2012?
?DomainName ?triton.ga? ?DomainNetbiosName ?TRITON? ?ForestMode ?Win2012?
?InstallDNS:$True ?LogPath ?C:\Windows\NTDS? ?NoRebootOnCompletion:$False
?SysVolPath ?C:\Windows\SysVol? ?force:$true }
```

1.14.2 Elevar o Nivel Funcional do Bosque

Os Niveis funcionais determinan as **capacidades** de dominio o bosque de Servicios de dominio de Active Directory (AD DS) que están dispoñibles. Tamén determinan os sistemas operativos Windows Server que se poden executar nos controladores de dominio do dominio ou do bosque.

> Por outro lado, os niveis funcionais NON afectan aos sistemas operativos que se poden executar nas estacións de traballo e nos servidores membros que están unidos ao dominio ou ao bosque.
> Cando implementamos AD DS, mellor establecer os niveis funcionais de dominio e de bosque no valor máis alto que admita a nosa contorna. Deste xeito, poderase empregar o maior número posible de características de AD DS.
> Por exemplo, e estamos seguros de que nunca se agregará ao dominio ou ao bosque controladores de dominio que executen Windows Server 2008, mellor seleccionar o nivel funcional de Windows Server 2012 (ou, do mesmo xeito Windows 2012 R2) durante o proceso de implementación. Polo contrario, se existe posibilidade de que se vaia conservar ou agregar algún controlador de dominio que execute Windows Server 2008, hai que seleccionar o nivel funcional de Windows Server 2008.

Para coñecer o Nivel Funcional no que se atopa o Bosque podemos empregar o comando Get-ADForest.

```
PS > $forest = Get-ADForest
PS > $forest.ForestMode
Windows2012R2Forest
```

Para elevar o Nivel Funcional do Bosque empregaremos o comando Set-ADForestMode. Vexamos un exemplo:

```
PS > Set-ADForestMode -Identity animamundi.ga -ForestMode Windows2012R2Forest
PS > Get-ADForest
ApplicationPartitions : {DC=ForestDnsZones,DC=animamundi,DC=ga, DC=DomainDnsZones,DC=triton,DC=ga}
CrossForestReferences : {}
DomainNamingMaster    : server01.triton.ga
Domains               : {triton.ga}
ForestMode            : Windows2012R2Forest
GlobalCatalogs        : {server01.triton.ga}
Name                  : triton.ga
PartitionsContainer   : CN=Partitions,CN=Configuration,DC=animamundi,DC=ga
RootDomain            : triton.ga
SchemaMaster          : server01.triton.ga
Sites                 : {Default-First-Site-Name}
SPNSuffixes           : {}
UPNSuffixes           : {}
```

1.14.3 Elevar o Nivel Funcional do Dominio

Faise do mesmo xeito que có Bosque:

```
PS > Set-ADDomainMode -Identity triton.ga -DomainMode Windows2012R2Domain

Confirmar
¿Está seguro de que desea realizar esta acción?
Se está realizando la operación "Set" en el destino "DC=triton,DC=ga".
```

```
[S] Sí  [O] Sí a todo  [N] No  [T] No a todo  [U] Suspender  [?] Ayuda (el valor predeterminado es "S"): S

PS > Get-ADDomain
AllowedDNSSuffixes                : {}
ChildDomains                      : {}
ComputersContainer                : CN=Computers,DC=triton,DC=ga
DeletedObjectsContainer           : CN=Deleted Objects,DC=triton,DC=ga
DistinguishedName                 : DC=triton,DC=ga
DNSRoot                           : triton.ga
DomainControllersContainer        : OU=Domain Controllers,DC=triton,DC=ga
DomainMode                        : Windows2012R2Domain
DomainSID                         : S-1-5-21-3472920172-2353984534-3407129034
ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=triton,DC=ga
Forest                            : triton.ga
InfrastructureMaster              : wserver1.triton.ga
LastLogonReplicationInterval      :
LinkedGroupPolicyObjects          : {CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=System,DC=triton,DC=
                                    ga}
LostAndFoundContainer             : CN=LostAndFound,DC=triton,DC=ga
ManagedBy                         :
Name                              : triton
NetBIOSName                       : triton
ObjectClass                       : domainDNS
ObjectGUID                        : 9026b492-72cb-4d8e-a3ff-b62f5c08c48d
ParentDomain                      :
PDCEmulator                       : wserver1.triton.ga
QuotasContainer                   : CN=NTDS Quotas,DC=triton,DC=ga
ReadOnlyReplicaDirectoryServers   : {}
ReplicaDirectoryServers           : {wserver1.triton.ga}
RIDMaster                         : wserver1.triton.ga
SubordinateReferences             : {DC=ForestDnsZones,DC=triton,DC=ga, DC=DomainDnsZones,DC=animamundi,DC=ga,
                                    CN=Configuration,DC=triton,DC=ga}
SystemsContainer                  : CN=System,DC=triton,DC=ga
UsersContainer                    : CN=Users,DC=triton,DC=ga
```

1.14.4 Integrar a zona DNS no Active Directory

Para realizar a integración da zona DNS no Active Directory seguir o seguinte enlace.
Para confirmar que un controlador de dominio foi rexistrado no DNS, hai que abrir a liña de comandos e executar:

```
PS> dcdiag /test:registerindns /dnsdomain:<domain name> /v
```

1.14.5 Automatizar a creación e configuración de obxectos do dominio

Active Directory ten alguhas ferramentas de liña de comandos **DS**. Os seguintes comandos **DS** son soportados en Windows Server 2008:

- Dsadd : Crea un obxecto no directorio activo.
- Dsget : Devolve atributos específicos dun obxecto.
- Dsmod : Modifica atributos específicos dun obxecto.
- Dsmove : Move un obxecto a un novo contedor ou Unidade Organizativa.
- Dsrm : Elimina un obxecto, todos os obxectos na subárbore baixo un obxecto contedor, ou ambos.
- Dsquery : Realiza unha consulta baseada nos parámetros proporcionados na liña de comandos e devolve unha lista cós obxectos que coinciden.

1.14.5.1 Crear Unidades Organizativas

- Para crear unidades organizativas empregamos o comando Dsadd OU.

```
PS C:\> dsadd ou "ou=PROGRAMACION, ou=TRITON00, dc=TRITON00, dc=LOCAL"
```

- O comando de Powershell comando para crealas:

```
PS C:\> New-ADOrganizationalUnit -DisplayName "Programacion" -Name "Programacion" -path "ou=TRITON00, dc=TRITON00, dc=LOCAL"
```

- Tamén podemos crear Unidades Organizativas empregando PowerShell.

```
# Conectámonos ao contedor da OU a crear
$objADSI = [ADSI]?LDAP://OU=TRITON00,DC=TRITON00,DC=LOCAL?

# Invocamos ao método Create do contedor indicando:
# - O tipo de obxecto a crear: "organizationalUnit"
# - O nome distintivo relativo, RDN: "PROGRAMACION"
$objOU = $objADSI.create("organizationalUnit", "OU=PROGRAMACION")

# Executamos os cambios en AD có método SetInfo do obxecto
$objOU.setInfo()
```

1.14.5.2 Crear usuarios

Vexamos distintas ferramentas para agregar usuarios:

- **Crear usuarios con Dsadd.**

O comando Dsadd user crea un obxecto de usuario e acepta parámetros que especifican as propiedades do usuario.

```
PS C:\> dsadd user "cn=Brais Piñeiro, ou=direccion, ou=triton00, dc=triton00, dc=local"
-samid brais -upn brais@triton00.local -fn Brais -mi BP -ln Piñeiro -display Brais -desc "Membro da Dirección"
-email brais@triton00.local -pwd abc123.. -mustchpwd yes
-hmdrv U: -hmdir \\servidor\usuarios$\brais -profile \\servidor\perfiles$\brais -loscr direccion.bat -disabled no
dsadd correcto:cn=Brais Piñeiro,ou=direccion,ou=triton00,dc=triton00,dc=local
PS C:\>
```

- O comando Dsadd crea o usuario na OU correspondente.
- Tamén lle configura os parámetros: Nome, Apelidos, Siglas, Descrición, email,...
- Configura o directorio persoal do usuario, pero ¡Ollo! **a utilidade non crea o directorio, ten ese erro!!!**. Para resolver este problema faremos do seguinte xeito:

```
# Créase a carpeta persoal do usuario
# PS> mkdir V:\usuarios\brais
PS> New-Item -ItemType directory -Path V:\usuarios\brais
# Rómpese a herdanza copiando os permisos existentes
# e logo dámoslle ao usuario Control Total sobre a súa carpeta
PS> icacls V:\usuarios\brais /inheritance:d /grant:r "brais:`(OI`)`(CI`)F"
```

- Configura o directorio do perfil móbil para o usuario.
- Configura un contrasinal base para o usuario e fai que o teña que cambiar no primeiro inicio de sesión.

- **Crear usuarios con PowerShell.**

Para crear usuarios empregando PowerShell podemos facelo de dous xeitos:

- O primeiro é empregar o cmdlet New-ADUser:

```
#Creamos un password cifrado
$password = ConvertTo-SecureString "abc123.." -AsPlainText -Force

#Crear el nuevo usuario
New-ADUser -Name "Profesor00" -SamAccountName "Profesor00" `
-GivenName "Profesor" -SurName "Torres" `
?path ?OU=profesores,OU=usuarios,OU=IES00,dc=IES00,dc=local? `
-Enabled $true -AccountPassword ${password} -ChangePasswordAtLogon $false `
-HomeDrive 'P:' -HomeDirectory '\\WSERVER1\Usuarios00$\Profesores00\profesor00' `
-ProfilePath '\\WSERVER1\Perfiles00$\profesor00' `
-ScriptPath 'inicio.bat'

##Crear directorio personal del usuario
#New-Item -Path 'V:\Usuarios00\Profesores00\' -Name 'Profesor00' -ItemType 'directory'
New-Item -Path '\\WSERVER1\Usuarios00$\Profesores00\' -Name 'Profesor00' -ItemType 'directory'

###Configurar permisos NTFS del directorio personal del usuario
####Administradores - CT
####Usuario dueño de ese directorio - CT
icacls '\\WSERVER1\Usuarios00$\Profesores00\Profesor00' /inheritance:r /grant:r 'Administradores:(OI)(CI)F'
icacls '\\WSERVER1\Usuarios00$\Profesores00\Profesor00' /grant:r 'Profesor00:(OI)(CI)F'
```

```
#Agregar el usuario a sus grupos
Add-ADGroupMember -Identity 'CN=G-Profesores,OU=profesores,OU=usuarios,OU=IES00,dc=IES00,dc=local' `
-Members 'CN=profesor00,OU=profesores,OU=usuarios,OU=IES00,dc=IES00,dc=local'
```

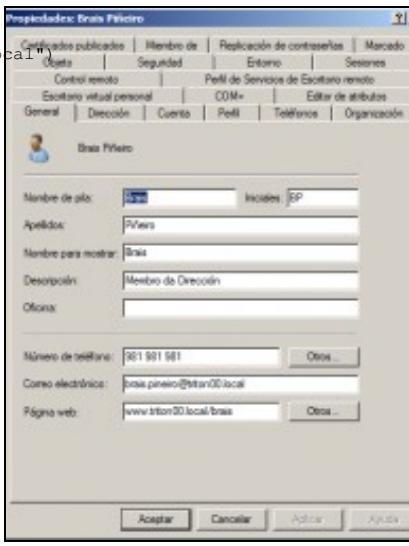
- O segundo xeito é o seguinte:

```
# Conectámonos ao contedor do usuario a crear
PS > $objOU=[ADSI]"LDAP://OU=direccion,ou=triton00,dc=triton00,dc=local"
# Invocamos ao método Create do contedor indicando:
# - O tipo de obxecto a crear: "user"
# - O nome distintivo relativo, RDN: "Andrea Torres"
PS > $objUser=$objOU.Create("user","CN=Andrea Torres")
# Configuramos ás propiedades que nos interesen do novo usuario
# para esta tarefa empregamos o método Put.
# - A única obrigatorio sAMAccountName :
PS > $objUser.Put("sAMAccountName","andreat")
# Executamos os cambios en AD có método SetInfo do obxecto
PS > $objUser.SetInfo()
```

Outras propiedades a configurar empregando o método **Put**:

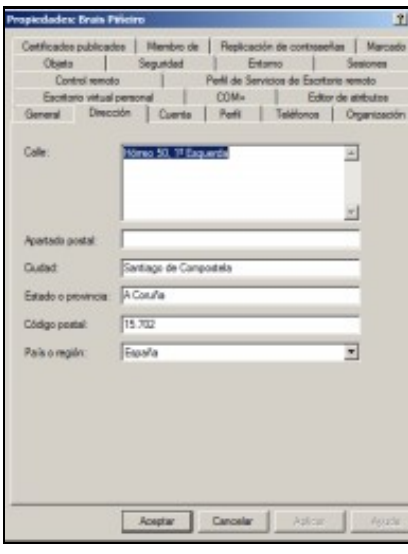
-Páxina onde se explican todas as propiedades dos Usuarios.
-É interesante empregar a ferramenta Editor ADSI (*Active Directory Service Interfaces*) para ver as propiedades configuradas de cada un dos obxectos do AD.

*** Ficha "General":**



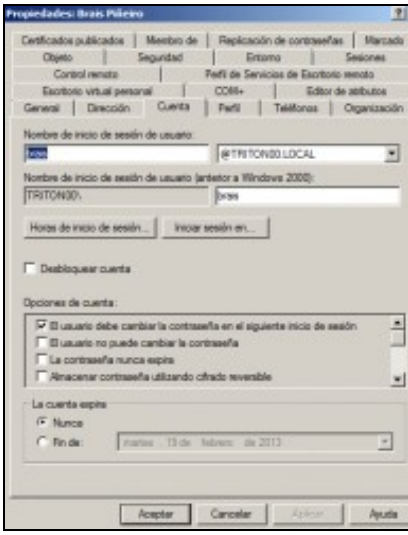

```
$objUser.Put("distinguishedName","CN=Brais Piñeiro,OU=Direccion,OU=Triton00,DC=Triton00,DC=Local")
$objUser.Put("cn","Brais Piñeiro")
$objUser.Put("name","Brais Piñeiro")
# Nombre de pila:
$objUser.Put("givenName","Brais")
# Iniciales:
$objUser.Put("initials","BP")
# Apellidos:
$objUser.Put("sn","Piñeiro")
# Nombre para mostrar:
$objUser.Put("displayName","Brais")
# Descripción:
$objUser.Put("description","Membro da Dirección")
# Número de Teléfono:
$objUser.Put("telephoneNumber","981 981 981")
# Correo electrónico:
$objUser.Put("mail","brais.pineiro@triton00.local")
# Página web:
$objUser.Put("wWWHomePage","www.triton00.local/brais")
```

*** Ficha "Dirección":**




```
# Calle:
$objUser.Put("streetAddress","Hórreo 50, 1º Esquerda")
# Ciudad:
$objUser.Put("l","Santiago de Compostela")
# Estado o provincia:
$objUser.Put("st","A Coruña")
# Código postal:
$objUser.Put("postalCode","15.702")
# Páis o región:
$objUser.Put("co","España")
$objUser.Put("c","ES")
$objUser.Put("countryCode","724")
```

- Táboa de códigos de países.

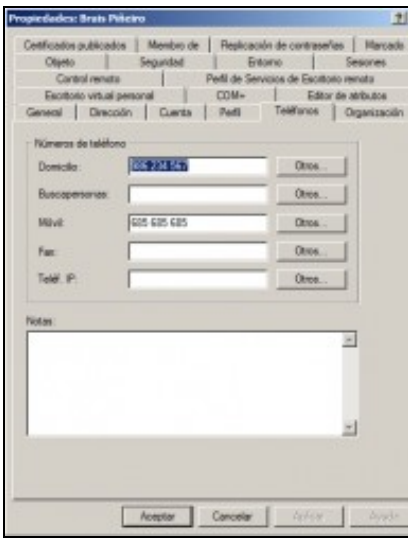


*** Ficha "Cuenta":**

```
# Nombre de inicio de sesión de usuario.
$objUser.Put("userPrincipalName","brais@triton00.local")
# Nombre de inicio de sesión de usuario (anterior a Windows 2000).
$objUser.Put("sAMAccountName","brais")
```

*** Ficha "Perfil":**

```
# Perfil de usuario.
# Ruta de acceso al perfil:
$objUser.Put("profilePath","\\servidor\perfiles$\brais")
# Script de inicio de sesión:
$objUser.Put("scriptPath","direccion.ps1")
# Carpeta particular.
# Conectar:
$objUser.Put("homeDrive","U:")
$objUser.Put("homeDirectory","\\servidor\usuarios$\brais")
```

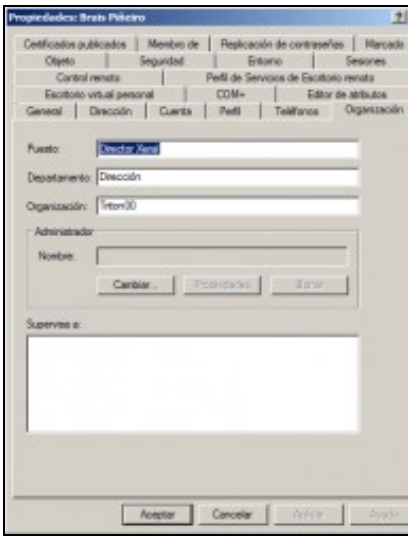


*** Ficha "Teléfonos":**

```
# Números de teléfono.
# Domicilio:
$objUser.Put("homePhone","986234567")
# Móvil:
$objUser.Put("mobile","685 685 685")
```

*** Ficha "Organización":**

```
# Puesto:
$objUser.Put("title","Director Xeral")
# Departamento:
$objUser.Put("departament","Dirección")
# Organización:
$objUser.Put("company","Triton00")
```



1.14.5.2.1 Administrar contas de usuario

O propósito principal dos obxectos de usuario en AD é dar soporte á autenticación dunha persoa ou dun servizo. As contas se crean, se administran e, as veces, se eliminan. Vexamos as tarefas administrativas máis comúns relacionadas coas contas de usuario:

- **Restabrecer o contrasinal de usuario:**

  Se o usuario esquece o seu contrasinal debemos cambiarlla por outra e logo configurar a conta para que o usuario teña que cambiar este novo contrasinal, pois o administrador non debería de sabela.
  - Para facer operación empregaremos o comando Dsmod:

```
PS C:\> dsmod user "cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local" -pwd abc123.. -mustchpwd yes
dsmod correcto:cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local
PS C:\>
```

  - Utilizando PowerShell, farémolo do seguinte xeito:

```
PS C:\> $objUser=[ADSI]"LDAP://cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local"
PS C:\> $objUser.setpassword("abc123..")
```

  Neste caso, a diferencia de outros atributos, non utiliza **SetInfo** despois de utilizar **SetPassword** para configurar o contrasinal do usuario. Pero, se queremos forzar ao usuario a modificar o contrasinal no seguinte inicio de sesión:

```
PS C:\> $objUser.Put("pwdLastSet",0)
PS C:\> $objUser.setinfo()
```

- **Desbloquear unha conta de usuario:**

  O bloqueo das contas faise por medio de Directivas, por exemplo, se un usuario non acerta varias veces o seu contrasinal.
  O único xeito de desbloquear unha conta de usuario é facer esta operación directamente na ferramenta gráfica, non hai xeito de facelo nin dende o símbolo de sistema nin con Windows PowerShell.

- **Deshabilitar e habilitar unha conta de usuario:**

  As contas de usuario son obxectos de seguridade moi importantes, polo tanto é importante:
  - Non deixar abertas contas de usuario.
  - Configurar directivas e auditorías de contrasinais, e procedementos para asegurarse de que as contas se están utilizando correctamente.
  - Deshabilitar a conta se un usuario se crea antes de que a necesite ou si este estará ausente por un longo período de tempo.

  Pódese deshabilitar unha conta dende o complemento *Usuarios e equipos de Active Directory* e tamén empregando o comando Dsmod:

```
PS C:\> dsmod user "cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local" -disabled yes
dsmod correcto:cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local
PS C:\>
```

  Para habilitar unha conta é cuestión de cambiar o **yes** por **no**.

Con Windows PowerShell temos dous xeitos de facelo:
- Empregando comandos:

```
# Deshabilitar:
PS> Disable-ADAccount ?Identity andrea
# Habilitar:
PS> Enable-ADAccount ?Identity andrea
```

- Ou indirectamente:

```
PS C:\> $objUser=[ADSI]"LDAP://cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local"
PS C:\> $objUser.psbase.InvokeSet('AccountDisabled',$true)
PS C:\> $objuser.setinfo()
```

Para habilitar unha conta é cuestión de cambiar o **$true** por **$false**.

- **Eliminar unha conta de usuario:**

Cando unha conta xa non é necesaria, pódese eliminar do AD.
- Pódense eliminar obxectos de AD empregando o comando Dsrm, simplemente do seguinte xeito:

```
PS C:\> Dsrm "cn=Andrea,ou=direccion,ou=triton00,dc=triton00,dc=local"
```

Observar que **Dsrm** non vai seguido pola clase de obxecto usuario (*user*) como nos outros comandos.
- Para eliminar un usuario de AD utilizando PS farémolo do seguinte xeito:

```
PS C:\> $objOU=[ADSI]"LDAP://ou=direccion,ou=triton00,dc=triton00,dc=local"
PS C:\> $objOU.Delete "user","CN=Andrea"
```

- **Mover unha conta de usuario:**

Moitas veces precisaremos mover un obxecto de usuario en AD dunha OU a outra.
- Pódense mover obxectos de AD empregando o comando Dsmove, simplemente do seguinte xeito:

```
PS C:\> dsmove "cn=andrea,ou=direccion,ou=triton00,dc=triton00,dc=local" -newparent "ou=programacion,ou=triton00,dc=trit
on00,dc=local"
dsmove correcto:cn=andrea,ou=direccion,ou=triton00,dc=triton00,dc=local
PS C:\>
```

Observar que **Dsmove** non especifica unha clase de obxecto de usuario.
- Para mover un usuario de AD utilizando PS farémolo do seguinte xeito:

```
PS C:\> $objUser=[ADSI]"LDAP://cn=andrea,ou=direccion,ou=triton00,dc=triton00,dc=local"
PS C:\> $objUser.psbase.MoveTo("LDAP://ou=programacion,ou=triton00,dc=triton00,dc=local")
```

- **Renomear unha conta de usuario:**

As veces nos interesa cambiar o nome de algún usuario, ademais de algún outro parámetro, vexamos como.
- Podemos empregar o comando [Dsmod] para cambiar algún atributo pero hai que ter en conta que non se poden modificar os atributos *samAccountName* e *CN*.
- En Windows PowerShell si podemos modificar a propiedade *CN*:

```
PS C:\> $objUser=[ADSI]"LDAP://cn=andrea,ou=direccion,ou=triton00,dc=triton00,dc=local"
PS C:\> $objUser.psbase.rename("cn=patricia")
```

Tamén se poden modificar outros atributos do nome, utilizando o método **Put** do obxecto de usuario.

1.14.5.2.2 Importar usuarios de xeito masivo

Nota: Se precisamos eliminar os acentos dun *string* podemos facelo coa función que se implementa no seguinte enlace.

- **Importar usuarios con CSVDE.**

CSVDE é unha ferramenta de liña de comandos que importa ou exporta obxectos de AD dende ou cara un arquivo de texto delimitado por comas (.csv). Estes arquivos pódense crear, modificar e abrir con ferramentas tales como o Bloc de Notas, o Calc ou o Excel. É un xeito moi sinxelo de automatizar a creación de contas de usuarios.

A sintaxe básica do comando é a seguinte:

csvde [-i] [-f nome de arquivos] [-k]
- O parámetro **i** especifica o modo de importación, pois, por defecto, o comando exporta.
- O parámetro **f** identifica o nome do arquivo que se importará ou exportará.
- O parámetro **k** é útil durante as operacións de importación, xa que fai que CSVDE ignore os erros como: *Obxecto xa existe, Violación de restrición e Atributo ou valor xa existen*.

Vexamos un exemplo dun arquivo típico .csv:

```
DN,objectClass,sAMAccountName,sn,givenName,userPrincipalName
"cn=Brais Piñeiro, ou=direccion, ou=triton00, dc=triton00, dc=local",user,brais,Brais,Piñeiro,brais@triton00.local
"cn=Andrea Torres, ou=direccion, ou=triton00, dc=triton00, dc=local",user,andrea,Andrea,Torres,andrea@triton00.local
```

Sendo:

· **DN** : Nome completo no AD.
· **objectClass** : Pódense importar tamén grupos.
· **sAMAccountName** : Nome de inicio de sesión de usuario (anterior a Windows 2000).
· **sn** : Nome de Pila.
· **givenName** : Apelidos.
· **userPrincipalName** : Nome de inicio de sesión de usuario.

O gran problema desta ferramenta é que non pode configurar un contrasinal inicial ao usuario e, polo tanto, estes usuarios están deshabilitados ata que non se lle configure unha. Un xeito de arranxar este problema é facer que o dominio admita contrasinais de **0 caracteres**, aquí ves cómo configurar así a Directiva de seguridade. Logo configuramos todas as novas contas para que pidan novo contrasinal no seguinte inicio de sesión, isto é fácil de facer simplemente accedendo á opción Propiedades seleccionando "todas" as contas que nos interesen "á vez".

- **Crear contas de usuario dende un .CSV empregando PowerShell:**

- Arquivo usuarios.csv

```
cn;nombre;apellidos;iniciales;ou;dn;dominio;descrip;password;changepwd;letraunidad;directperson;directperfil;scriptinicio;deshabilit
BraisP;Brais;Piñeiro;BP;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=BraisP,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAG
MariaR;María;Rodríguez;MR;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=MariaR,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTI
SoniaL;Sonia;López;SL;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=SoniaL,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,
MilagrosF;Milagros;Fernández;MF;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=MilagrosF,ou=ALUMNOS,ou=USUARIOS,ou
NoeF;Noé;Fouz;NF;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=NoeF,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESS
BreixoG;Breixo;Gómez;BG;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=BreixoG,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIA
LauraL;Laura;López;LL;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=LauraL,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,
NuriaL;Nuria;López;NL;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=NuriaL,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,
AndreaP;Andrea;Pin;AP;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=AndreaP,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO
RocioP;Rocío;Polín;RP;ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,dc=IESSANTIAGO,dc=LOCAL;cn=RocioP,ou=ALUMNOS,ou=USUARIOS,ou=IESSANTIAGO,
```

- Script PowerShell:

```
#Dar de alta de forma masiva a Alumnos
#Cabecera de las filas:
###cn;nombre;apellidos;iniciales;ou;dn;dominio;descrip;password;changepwd;
###letraunidad;directperson;directperfil;scriptinicio;deshabilitada;grupoprin

$usuarios = Import-Csv C:\scripts\usuarios.csv -Delimiter ";"

foreach ($usuario in $usuarios) {

   #Damos de alta al usuario con el comando dsadd
   dsadd user ${usuario}.dn -samid ${usuario}.cn -upn "$(${usuario}.cn)@$(${usuario}.dominio)" `
   -fn ${usuario}.nombre -mi ${usuario}.iniciales -ln ${usuario}.apellidos -display ${usuario}.cn `
   -desc ${usuario}.descrip -email "$(${usuario}.cn)@$(${usuario}.dominio)" `
   -pwd ${usuario}.password -mustchpwd ${usuario}.changepwd `
   -hmdrv ${usuario}.letraunidad -hmdir ${usuario}.directperson -profile ${usuario}.directperfil `
```

```
    -loscr ${usuario}.scriptinicio -disabled ${usuario}.deshabilitada `

    #Creamos el directorio personal del usuario
    New-Item -ItemType directory -Path ${usuario}.directperson

    #Configuramos los permisos del directorio personal del usuario
    icacls ${usuario}.directperson /inheritance:d /grant:r "$(${usuario}.cn):(OI)(CI)F"

    #Agregamos el usuario a su grupo principal
    dsmod group ${usuario}.grupoprin -addmbr ${usuario}.dn
}
```

1.14.5.3 Grupos

- **Alcance dos grupos**

Para ofrecer máis seguridade temos no AD varios grupos que teñen distinto "alcance".
O ámbito dun grupo determinado indica que obxectos poden conter ese grupo, limitando a obxectos do mesmo dominio ou permitindo obxectos de outros dominios. Tamén controla onde pode ser utilizado ese grupo, no dominio ou en calquera dominio do bosque.

- **Grupos Locais do Dominio**

Grupos Locais do Dominio poden ter no seu interior os seguintes membros:
- Usuarios
- Equipos
- Grupos Globais de calquera Dominio do Bosque
- Grupos Universais
- Grupos Locais do mesmo Dominio

Así, os grupos locais empréganse para asignar permisos a recursos no mesmo dominio ao que pertence ese grupo local.

- **Grupos Globais**

Os Grupos Globais poden ter os seguintes membros:
- Usuarios
- Equipos
- Outros Grupos Globais do mesmo dominio

Pódense empregar Grupos Globais para configurar permisos sobre recursos localizados en calquera dominio dun bosque. Isto pódese conseguir engadindo o grupo global como membro dun grupo local que teña os permisos requiridos.
Un grupo global pode pertencer a outro grupo global só se os dous están no mesmo dominio.

- **Grupos Universais**

Os Grupos Universais poden ter calquera dos seguintes tipos de membros:
- Usuarios
- Equipos
- Grupos Globais de calquera dominio do Bosque
- Outros Grupos Universais

Tendo esto en conta, podemos configurar o acceso a obxectos dun dominio do bosque empregando grupos locais facendo que os universais se aniden nestes.
Pódense empregar tamén grupos universais para configurar permisos a grupos e contas que, ou ben abarcan varios dominios ou todo o bosque.
Un punto importante no emprego dos grupos universais é que a pertenencia a eles non debe cambiar a menudo, pois os grupos universais son almacenados no catálogo global. Os cambios de usuarios e grupos pertencentes ao grupo universal son replicados a todos os servidores de catálogo global de bosque. Se estes cambios se realizan a menudo poden consumir moito ancho de banda.

- **Anidamento de grupos**

Como discutimos antes, o anidamento de grupos é o feito de agregar un grupo como membro de outro grupo. Por exemplo, cando facemos un grupo global como membro dun grupo universal, dise que se fixo un anidamento a un grupo universal.
O anidamento de grupos reduce o número de veces que hai que asignar permisos a usuarios en diferentes dominios dun bosque. Por exemplo, se temos múltiples dominios fillos no noso AD, e existen usuarios de cada dominio que precisan acceder a unha base de datos localizada no dominio pai, o xeito máis sinxelo de facelo sería o seguinte:

1. Crear un grupo global en cada dominio que conteña todos os usuarios que precisan acceder á base de datos.
2. Crear un grupo universal no dominio pai. E logo poñer como membros deste dominio os grupos globais antes creados.
3. Agregar o grupo universal a un grupo local para asignarlle os permisos necesarios para acceder á base de datos.

   Resumindo, hai que empregar o método: Añadir os **Contas** a grupos **Globais**, añadir os grupos globais a grupos **Universais**, añadir os grupos universais a grupos **Locais** e, finalmente, asignar permisos a eses grupos locais.

   No caso que máis nos imos atopar, que é, un único dominio nun bosque, o normal é empregar Grupos Globais de Seguridade anidados e, os permisos, directamente aplicados a estes Grupos Globais.

1.14.5.3.1 Crear grupos

- Para a creación de grupos empregaremos o comando Dsadd group tal e como podemos ver a continuación:

```
PS C:\> dsadd group "cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL" -scope g
dsadd correcto:cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL
```

  Facer o anidamento de grupos e introducir os membros deste grupo no seu interior, ainda que tamén se pode facer con este comando, deixarémolo mellor para os comandos Dsmod.

Outro exemplo, pero empregadno os cmdlets **New-ADGroup** e **Set-ADGroup**.

```
PS> New-ADGroup -Name "G-Usuarios" -SamAccountName G-Usuarios -GroupCategory Security -GroupScope Global -DisplayName "G-Usuarios" -P
PS> New-ADGroup -Name "G-Directores" -SamAccountName G-Directores -GroupCategory Security -GroupScope Global -DisplayName "G-Director
PS> New-ADGroup -Name "G-Tecnicos" -SamAccountName G-Tecnicos -GroupCategory Security -GroupScope Global -DisplayName "G-Tecnicos" -P
PS> New-ADGroup -Name "G-Programadores" -SamAccountName G-Programadores -GroupCategory Security -GroupScope Global -DisplayName "G-Pr
PS> Add-ADGroupMember -Identity G-Usuarios -Members G-Directores,G-Tecnicos,G-Programadores
```

- Se queremos buscar todos os grupos que comezen por 'G-':

```
PS>Get-ADGroup  -Filter {name -like "G-*"}
```

- Tamén podemos facelo con PowerShell:

```
# Conectámonos ao contedor do usuario a crear
PS > $objOU=[ADSI]"LDAP://OU=direccion,ou=triton00,dc=triton00,dc=local"
# Invocamos ao método Create do contedor indicando:
# - O tipo de obxecto a crear: "group"
# - O nome distintivo relativo, RDN: "G-Direccion"
PS > $objGroup=$objOU.Create("group","G-Direccion")
# Configuramos ás propiedades que nos interesen do novo grupo
# para esta tarefa empregamos o método Put.
# - A única obrigatorio sAMAccountName :
PS > $objGroup.Put("sAMAccountName","G-Direccion")
PS >
# Executamos os cambios en AD có método SetInfo do obxecto
PS > $objGroup.SetInfo()
```

  - Por defecto créase un Grupo de Seguridade Global, se quixeramos especificalo:

```
$objGroup.Put("groupType", 0x80000002)
```

  - Se queremos configuralo como Grupo de Seguridade Local:

```
$objGroup.Put("groupType", 0x80000004)
```

- Por último, tamén podemos importar grupos con **CSVDE**.

  Primeiramente creamos un arquivo .csv que chamaremos **grupos.csv**:

```
objectClass, sAMAccountName,DN,member
group,G-Direccion,"cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL",
"cn=Andrea,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL;
cn=Brais,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL"
...
```

Logo, dende a liña de comandos:

```
PS >csvde -i -f c:\scripts\grupos.csv
```

1.14.5.3.2 Anidar grupos e agregarlle membros

Para estas tarefas empregaremos o comando Dsmod group, vexamos algún exemplo interesante:

- **Anidar grupos:**

```
# G-DIRECCION será membro de G-TRITON
PS C:\> dsmod group "cn=G-TRITON,ou=TRITON00,dc=TRITON00,dc=LOCAL"
-addmbr "cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL"

dsmod correcto:cn=G-TRITON,ou=TRITON00,dc=TRITON00,dc=LOCAL
PS C:\>
```

- **Agregar varios membros a un grupo existente:**

```
# andrea e Brais Piñeiro serán membros de G-DIRECCION
PS C:\> dsmod group "cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL"
-addmbr "cn=andrea,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL"
"cn=Brais Piñeiro,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL"

dsmod correcto:cn=G-DIRECCION,ou=DIRECCION,ou=TRITON00,dc=TRITON00,dc=LOCAL
```

- **Agregar membros a un grupo con PowerShell:**

Deste xeito que imos ver podemos anidar grupos ou administrar a pertenencia de usuarios en grupos:

```
#Creamos los grupos
PS > New-ADGroup -Name "G-Usuarios" -SamAccountName G-Usuarios -GroupCategory Security -GroupScope Global -DisplayName "G-Usuarios" -
PS > New-ADGroup -Name "G-Directores" -SamAccountName G-Directores -GroupCategory Security -GroupScope Global -DisplayName "G-Directo
PS > New-ADGroup -Name "G-Programadores" -SamAccountName G-Programadores -GroupCategory Security -GroupScope Global -DisplayName "G-P
PS >  New-ADGroup -Name "G-Tecnicos" -SamAccountName G-Tecnicos -GroupCategory Security -GroupScope Global -DisplayName "G-Tecnicos"
#Anidamiento
PS >  Add-ADGroupMember -Identity G-Usuarios -Members G-Directores,G-Programadores,G-Tecnicos
#Comprobamos anidamiento
PS > Get-ADGroupMember -Identity G-Usuarios | select distinguishedname

distinguishedname
-----------------
CN=G-Directores,OU=Directores,OU=Usuarios,OU=SL,DC=sl,DC=local
CN=G-Programadores,OU=Programadores,OU=Usuarios,OU=SL,DC=sl,DC=local
CN=G-Tecnicos,OU=Tecnicos,OU=Usuarios,OU=SL,DC=sl,DC=local
```

Para eliminar membros dun grupo faise o mesmo pero se emprega o método *Remove* en vez do método *Add*.

- *Descubrir os Grupos que comecen por "G-":*

```
PS>Get-ADGroup  -Filter {name -like "G-*"}
```

- **Descubrir os membros dun grupo:**

Empregaremos o cmdlet Get-ADGroupMember:

```
PS>Get-ADGroupMember -Identity "G-Usurios"
...
```

- **Exemplo onde se monta un directorio en concreto se ese usuario pertence a un grupo determinado:**

Supoñer a situación de que si un usuario que inicia sesión nun equipo cliente pertence ao grupo **G-ADMINISTRACION** terá montada na unidade **W:** o recurso compartido **wserver****facturas$**:

```
# Se o usuario que inicia pertence a G-ADMINISTRACION
# montámoslle o directorio \\wserver\facturas$:
$servidor=wserver
```

```
$administ="CN=G-ADMINISTRACION,OU=ADMINISTRACION,OU=USUARIOS,OU=TRITON,DC=TRITON,DC=LOCAL"
$usuario = $env:username
$grupos =  (Get-ADUser -identity $usuario -Properties memberof | Select-Object memberof).memberof
foreach ($grupo in $grupos)
{
if ($grupo -eq $administ)
{
net use W: \\$servidor\facturas$ /persistent:no
}

}
```

Recorda que, para que este *script* funcione no Cliente Windows 8.1:
_ O arquivo inicio.ps1 gardarase no recurso compartido \\wserver\netlogon
_ Hai que configurar a *Set-ExecutionPolicy* do Windows 8.1 como *RemoteSigned* para que poida executar arquivos de PowerShell.
_ Hai que asociar os arquivos de extensión .ps1 ao PowerShell empregando a Directiva de grupo correspondente.
_ Hai que instalar Herramientas de administración remota del servidor para Windows 8,1 para poder empregar o comando *Get-ADUser* do *script*.

- Herramientas de administración remota del servidor para Windows 10.

1.14.5.3.3 Mover, Renombrar e Eliminar grupos

Para mover ou cambiar o nome dun obxecto podemos empregar o comando **Dsmove**.

- **Para cambiar o nome do grupo G-Instaladores a G-AdminSoft:**

```
PS >dsmove "cn=G-Instaladores,ou=triton00,dc=triton00,dc=local" -newname "G-AdminSoft"
```

- **Mover un grupo a unha nova ubicación:**

```
PS >dsmove "cn=G-AdminSoft,ou=triton00,dc=triton00,dc=local" -newparent "ou=Administradores,ou=triton00,dc=triton00,dc=local"
```

- **Para eliminar grupos:**

Pódese utilizar para esta tarefa o comando **Dsrm** para eliminar un grupo (ou calquera outro obxecto de AD):

```
PS >dsrm "cn=G-AdminSoft,ou=Administradores,ou=triton00,dc=triton00,dc=local"
```

1.14.5.4 Exemplo de creación dun usuario

Exemplo de creación do usuario PIA. Características:

- Unidade Organizativa: "ou=PROFESORES, ou=USUARIOS, ou=TRITON, dc=TRITON, dc=GA"
- Conta: "pia"
- Mail: "pia@triton.ga"
- Apelidos: "Pérez"
- Descrición: "Profesor@"
- Contrasinal: "abc123.." - Ten que cambialo no seguinte inicio de sesión.
- Carpeta persoal: "\wserver1\usuarios$\profesores\pia" conectada na unidade de rede U:
- Perfil móbil: "\\wserver1\perfiles$\pia"
- Script de inicio de sesión: "inicio.ps1"
- Grupo: "cn=G-PROFESORES, ou=PROFESORES, ou=USUARIOS, ou=TRITON,dc=TRITON,dc=GA"

Solución:

```
PS> dsadd user "cn=Pia, ou=PROFESORES, ou=USUARIOS, ou=ANIMAMUNDI, dc=ANIMAMUNDI, dc=GA"
-samid pia -upn pia@animamundi.ga -fn Pia -mi PP -ln Pérez -display Pia
-desc "Profesor@" -email pia@gmail.com -pwd abc123.. -mustchpwd yes
-hmdrv U: -hmdir \\wserver1\usuarios$\profesores\pia
-profile \\wserver1\perfiles$\pia -loscr inicio.ps1 -disabled no
PS> mkdir V:\usuarios\profesores\pia
PS> icacls V:\usuarios\profesores\pia /inheritance:d /grant:r "pia:`(OI`)`(CI`)F"
PS> dsmod group "cn=G-PROFESORES, ou=PROFESORES, ou=USUARIOS, ou=ANIMAMUNDI,dc=ANIMAMUNDI,dc=GA"
-addmbr "cn=pia, ou=PROFESORES, ou=USUARIOS, ou=ANIMAMUNDI, dc=ANIMAMUNDI,dc=GA"
```

1.14.5.5 Administrar equipos

1.14.5.5.1 Crear equipos no AD

- A utilidade Dsadd.exe permítenos crear Equipos dende a liña de comandos, tal e como se fixo antes cós usuaros. Pódese crear un script empregando esta ferramenta.

  A sintaxe básica é a seguinte:

```
PS> dsadd computer WCLIENT00
```

- Windows PowerShell tamén inclúe o cmdlet New-ADComputer, que nos permite crear equipos coa mesma sintaxe básica. Ollo, pois estas ferramentas crean o boxecto equipo pero non meten o equipo no dominio.

```
PS> New-ADComputer -Name "WCLIENT00" -SamAccountName "WCLIENT00" -Path "OU=Clientes,OU=Equipos,OU=IES,DC=IES,DC=LOCAL"
```

1.14.5.5.2 Agregar equipos ao dominio

Temos distintos xeitos:

- Empregar a utilidade de liña de comandos netdom.exe:

```
PS> netdom join <computername> /Domain:<DomainName> [/UserD:<User> /PasswordD:<UserPassword>] [/OU:OUDN]
```

- Agregar equipos polos usuarios de xeito indirecto grazas á *Default Domain Controlers Policy Group Policy Object* (GPO) que permite a calquera usuario do dominio agregar ata 10 equipos. Se queremos que un usuario en especial (neste caso "equipator") poida engadir máis de 10 equipos podemos delegar nel ese rol.
- Un script para engadir os equipos no dominio sería o seguinte:

```
#Hai que adaptar as seguintes variables
$usuario = "equipator"
$contrasinal = "abc123.."
$dominio = "ies.local"

$password = ConvertTo-SecureString $contrasinal -AsPlainText -Force
$username = $dominio + "\" + $usuario
$myCred = New-Object System.Management.Automation.PSCredential $username, $password
Add-Computer -DomainName  $dominio -Credential $myCred -OUPath "OU=Clientes,OU=Equipos,OU=IES,DC=IES,DC=LOCAL"

#Reiniciamos o equipo
Restart-Computer
```

1.14.5.5.3 Buscar y mover equipos

```
PS> dsquery computer -name wclient00
"CN=WCLIENT00,CN=Computers,DC=ANIMAMUNDI,DC=GA"
PS> dsmove "CN=WCLIENT00,CN=COMPUTERS,DC=ANIMAMUNDI,DC=GA"
-newparent "OU=CLIENTES,OU=EQUIPOS,OU=ANIMAMUNDI,DC=ANIMAMUNDI,DC=GA"
dsmove correcto:CN=WCLIENT00,CN=COMPUTERS,DC=ANIMAMUNDI,DC=GA
PS V:\>
```

1.14.5.5.4 Habilitar e deshabilitar equipos

```
# Deshabilitar:
PS> Disable-ADAccount ?Identity wclient00
# Habilitar:
PS> Enable-ADAccount ?Identity wclient00
```

1.14.6 Buscar elementos do AD

Podemos buscar elementos no AD de distintos xeitos, como coas "Consultas gardadas" e o "Botón Atopar da Barra de Ferramentas" do complemento "Usuarios e equipos de AD".
Có comando Dsquery poderemos atopar obxectos de AD.

- O mellor con estes comandos é acudir á axuda:

```
PS C:\> dsquery /?
Descripción: el conjunto de comandos de esta herramienta le permite consultar
el directorio de acuerdo con los criterios especificados. Cada uno de los
siguientes comandos de dsquery busca objetos de un tipo específico, excepto
dsquery *, que puede consultar cualquier tipo de objetos.

dsquery computer  - busca equipos en el directorio.
dsquery contact   - busca contactos en el directorio.
dsquery subnet    - busca subredes en el directorio.
dsquery group     - busca grupos en el directorio.
dsquery ou        - busca unidades organizativas en el directorio.
dsquery site      - busca sitios en el directorio.
dsquery server    - busca instancias de DC/LDS de Active Directory en el
                    directorio.
dsquery user      - busca usuarios en el directorio.
dsquery quota     - busca especificaciones de cuota en el directorio.
dsquery partition - busca particiones en el directorio.
dsquery *         - busca cualquier objeto en el directorio con una consulta
                    genérica de LDAP.

Para obtener ayuda sobre un comando específico, escriba
"dsquery <tipoObjeto>/?", donde <tipoObjeto> es uno de los tipos de objeto
compatibles mostrados más arriba. Por ejemplo, dsquery ou /?.
...
```

- Logo de especificar o tipo de obxecto, pódense utilizar parámetros para indicar criterios da consulta:

  · Polo seu nome: **-name**
  · Pola súa descrición: **-desc**
  Para ver as propiedades que se requiren:

```
PS C:\> dsquery user /?
Descripción: busca los usuarios en el directorio que coincidan con
ciertos criterios.

Sintaxis:   dsquery user [{<nodoInicio> | forestroot | domainroot}]
            [-o {dn | rdn | upn | samid}]
            [-scope {subtree | onelevel | base}]
            [-name <nombre>] [-namep <nombreFonético>]
            [-desc <descripción>] [-upn <UPN>]
            [-samid <nombreSAM>] [-inactive <númSemanas>] [-stalepwd <númDías>]
            [-disabled] [{-s <servidor> | -d <dominio>}] [-u <nombreUsuario>]
            [-p {<contraseña> | *}] [-q] [-gc] [-limit <númObjetos>]
            [{-uc | -uco | -uci}]
```

  Por exemplo, para buscar todos os usuarios que comezan por "b":

```
PS C:\> dsquery user -name b*
"CN=Brais Piñeiro,OU=DIRECCION,OU=TRITON00,DC=TRITON00,DC=LOCAL"
```

  E, se non queremos ver a saída polo seu DN, senón polo seu nome de "inicio de sesión":

```
PS C:\> dsquery user -name b* -o upn
"brais@triton00.local"
```

1.14.6.1 Descubrir todos os Grupos existentes dentro dunha UO

```
PS > Get-ADGroup -filter * -Properties Distinguishedname |
Select-Object distinguishedname | where {$_ -match "OU=TRITON"}

distinguishedname
-----------------
CN=G-USUARIOS,OU=USUARIOS,OU=TRITON,DC=triton,DC=ga
```

1.14.6.2 Descubrir todos os Usuarios existentes dentro dunha UO

```
PS > Get-ADUser -filter * -Properties Distinguishedname |
Select-Object distinguishedname | where {$_ -match "OU=TRITON"}

distinguishedname
-----------------
CN=andrea,OU=USUARIOS,OU=TRITON,DC=triton,DC=ga
```

1.15 Enlaces Interesantes

- ◊ PowerShell en PowerShell.com
- ◊ PowerShell en Microsoft.com
- ◊ ...