|  |
| --- |
| Workshop de ALM - Laboratorio 01 |
| Despligue |
|  |

|  |
| --- |
|  |

## Introducción

Uno de los puntos más delicados a la hora de abordar cualquier proyecto de SharePoint ya sea Online u OnPremise es abordar el despliegue de los artefactos propios del sistema. Estos artefactos son Columnas de Sitio, Tipos de Contenido, Páginas, Sitios, Listas, Biblioteca de Documentos, etc...

En un entorno OnPremise existen varias formas de hacerlo bien mediante Visual Studio (ya sea utilizando los XML (que en un principio desaconsejaron) o bien a través de las APIs de SharePoint) o bien haciendo uso de PowerShell.

Para entornos en la nube podemos utilizar la API pero mediante PowerShell tenemos la limitación de que los cmdlets disponibles. En estas circunstancias una serie de Gurus de Office 365 junto con la colaboración de Microsoft Corporation crearon un proyecto Open Source llamado Office 365 Patterns and Practice (a partir de ahora PNP) cuya finalidad es extender todas las características disponibles en la plataforma y unificar la forma en la que trabajan todos los desarrolladores de Office. Este proyecto está compuesto por diversas soluciones que se pueden utilizar tanto en PowerShell como en .NET. En este laboratorio vamos a ver como podemos utilizar PNP para facilitarnos nuestros despliegues

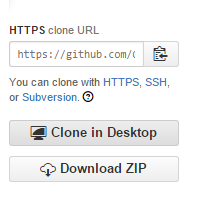
# Laboratorio 1 Introducción a PNP

Requisitos Previos:

1. Instalarse este paquete para abrir el proyecto de Soluciones <http://wix.codeplex.com>.
2. Instalar el modúlo de PowerShell 4.0

**Manos a la obra:**

1. Ir a la siguiente dirección <https://github.com/OfficeDev/PnP> y descargarse el proyecto fuente PNP para ello bien lo podemos hacer desde Visual Studio mediante GIT o descargarnos el código desde la propia interfaz.



1. A continuación abrimos una Consola PowerShell y nos dirigimos a la carpeta Solutions\PowerShell.Commands. Para poder ejecutar estos comandos desde nuestra consola PowerShell debemos de ejecutar el script Install.ps1 que lo que hace es desplegar los módulos disponibles dentro la carpeta de modulos de PowerShell.
2. Algunos ejemplos de cmlets disponibles son los siguientes :
   1. Acceso al Site

Connect-SPOnline -Url https://yourtenant.sharepoint.com

# Connecting with specified credentials

$creds = Get-Credential

Connect-SPOnline -Url https://yourtenant.sharepoint.com -Credentials $creds

# Connecting with credentials that are stored in the credential manager. Connect-SPOnline -Url https://yourtenant.sharepoint.com -Credentials "LABEL"

* 1. Consulta de listas:

$siteUrl = "https://<yourtenant>.sharepoint.com"

$credentials = Get-Credential

Connect-SPOnline -Url $siteUrl -Credentials $credentials

$ctx = Get-SPOContext

$w = $ctx.Web

$w.Lists.GetByTitle("TestList")

$ctx.Load($w)

Execute-SPOQuery # Or use $ctx.ExecuteQuery()

* 1. Aprovisionar Site

$basePath = "C:\DeploymentFiles"

$themePath = "/sites/psdemo/\_catalogs/theme/15" # ServerRelativeUrl

$tenant = "<yourtenantname>"

$tenantAdmin = Get-Credential -Message "Enter Tenant Administrator Credentials"

Connect-SPOnline -Url https://<tenant>-admin.sharepoint.com -Credentials $tenantAdmin

New-SPOTenantSite -Title "PS Site" -Url "https://$tenant.sharepoint.com/sites/psdemo" -Owner $tenantAdmin -Lcid 1033 -TimeZone 24 -Template STS#0 -RemoveDeletedSite -Wait

# Connect with the tenant admin credentials to the newly created site collection

Connect-SPOnline -Url https://$tenant.sharepoint.com/sites/psdemo -Credentials $tenantAdmin

# Set Property Bag key to designate the type of site you're creating

Set-SPOPropertyBagValue -Key "PNP\_SiteType" -Value "PROJECT"

# Upload a theme

Add-SPOFile -Path "$basePath\contoso.spcolor" -Url "$themePath/contoso.spcolor"

Add-SPOFile -Path "$basePath\contoso.spfont" -Url "$themePath/contoso.spfont"

Add-SPOFile -Path "$basePath\contosobg.jpg" -Url "$themePath/contosobg.jpg"

Set-SPOTheme -ColorPaletteUrl "$themePath/contoso.spcolor" -FontSchemeUrl "$themePath/contoso.spfont" -BackgroundImageUrl "$themePath/contosobg.jpg"

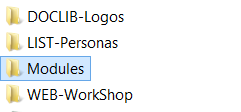
# Add a list and add a field to the list.

New-SPOList -Title "Projects" -Template GenericList -Url "lists/projects" -QuickLaunchOptions on

Add-SPOField -List "Projects" -InternalName "ProjectManager" -DisplayName "Project Manager" -StaticName "ProjectManager" -Type User -AddToDefaultView

# Laboratorio 2 Creando una Estructura de Despliegue

En este laboratorio vamos a extender el proyecto PNP para facilitarnos el despliegue de nuestras aplicaciones. Para ello vamos en primer lugar a crearnos una estructura en la que vamos a distribuir los artefactos que queremos desplegar. Por convención nos crearemos una carpeta que empiece por WEB para definirnos los subsitios DOCLIB para las bibliotecas de documentos y LIB para las bibliotecas personalizadas. En nuestro caso tendrá la siguiente estructura:



Dentro de cada carpeta nos definiremos un fichero manifest.xml en el que especificaremos los valores que va a tener este artefacto.

Para la Biblioteca de Documentos el manifest sería el siguiente:

<?xml version="1.0" encoding="utf-8"?>

<List >

<Url>Logos</Url>

<Name>Logos</Name>

<Description>Logos que se van a visualizar en los enlaces configurables.</Description>

</List>

Para una Lista el manifest sería el siguiente:

<?xml version="1.0" encoding="utf-8"?>

<List>

<Url>Persona</Url>

<Name>Persona</Name>

<Description>Lista de los personas que vienen al Workshop</Description>

<ContentTypes>

<ContentType LCID="3082" Name="Persona CEUS" />

</ContentTypes>

</List>

Para el Web el Manifest sería el siguiente:

<?xml version="1.0" encoding="utf-8"?>

<Site>

<RelativeUrl>/WorkShop</RelativeUrl>

<Name>WorkShop</Name>

<Description>Workshop</Description>

<Template>STS#0</Template>

<Language>3082</Language>

<SiteMasterPage>/\_catalogs/masterpage/seattle.master</SiteMasterPage>

<SystemMasterPage>/\_catalogs/masterpage/seatlle.master</SystemMasterPage>

</Site>

A continuación nos crearemos un fichero ColumnSite.xml, para declararlos las columnas de sitio que vamos a utilizar. Este fichero tiene la siguiente definición:

<?xml version="1.0" encoding="utf-8"?>

<Site>

<SiteColumns>

<Add

Id="009246d9-1d2f-49e0-933d-b33ce03c1a81"

FieldType="DateTime"

Name="CEUSFechaNacimiento"

DisplayName="Fecha de nacimiento"

StaticName="CEUSFechaNacimiento"

Group="WorkShop Ceus SiteColumns"

Hidden="False"

Required="True"

Sealed="False"

ShowInDisplayForm="True"

ShowInEditForm="True"

ShowInListSettings="True"

ShowInNewForm="True">

<DisplayNames>

<DisplayName Label="es-ES" Value="Fecha de nacimiento" />

</DisplayNames>

</Add>

<Add

Id="A26949E9-1F69-43CB-9EA3-880F451A70BE"

FieldType="Text"

Name="CEUSNombre"

DisplayName="Nombre"

StaticName="CEUSNombre"

Group="WorkShop Ceus SiteColumns"

Hidden="False"

Required="False"

Sealed="False"

ShowInDisplayForm="True"

ShowInEditForm="True"

ShowInListSettings="True"

ShowInNewForm="True">

<DisplayNames>

<DisplayName Label="es-ES" Value="Nombre" />

</DisplayNames>

</Add>

<Add

Id="2EF1BE2D-B02B-4782-84C5-729CE5F5D8DF"

FieldType="Text"

Name="CEUSApellidos"

DisplayName="Apellidos"

StaticName="CEUSApellidos"

Group="WorkShop Ceus SiteColumns"

Hidden="False"

Required="False"

Sealed="False"

ShowInDisplayForm="True"

ShowInEditForm="True"

ShowInListSettings="True"

ShowInNewForm="True">

<DisplayNames>

<DisplayName Label="es-ES" Value="Nombre" />

</DisplayNames>

</Add>

</SiteColumns>

</Site>

Una vez tenemos creadas las columnas de Sitio nos crearemos un Tipo de Contenido, para ello nos definimos la siguiente nomenclatura: ContentTypes-v{XX.XX}-{Nombre del Content Type} Donde XX.XX es la versión de este ContentType. En nuestro caso vamos a crearnos Tipo de Contenido Persona con la siguiente definición:

<?xml version="1.0" encoding="utf-8"?>

<Site>

<ContentTypes>

<Add

Id="0x0100655593407AD04FE0B1B77FCBCD12806B"

Parent="Item"

Name="Persona CEUS"

DisplayName="Persona CEUS"

StaticName="Persona CEUS"

Group="WorkShop Ceus Content Types"

Description="">

<DisplayNames>

<DisplayName Label="es-ES" Value="Persona CEUS" />

</DisplayNames>

<Fields>

<Add InternalName="CEUSFechaNacimiento" />

<Add InternalName="CEUSNombre" />

<Add InternalName="CEUSApellidos" />

</Fields>

</Add>

</ContentTypes>

</Site>

Una vez tengamos la estructura vamos a utilizar PNP para que realice el funcionamiento que deseamos que es crearnos todos estos elementos.

En primer lugar nos vamos a crear un módulo con todas las funciones para trabajar con Columnas de sitio.

<#

Crea la columna especificada en el XML. Si ya existe la actualiza con la nueva información

#>

Function CreateOrUpdateFieldFromXML {

param ($FieldXML, $Context)

$FieldQuery = Get-SPOField -Identity $FieldXML.Name -ErrorAction SilentlyContinue

Execute-SPOQuery

if($FieldQuery.Id -eq $null) {

CreateField -FieldXML $FieldXML -Context $Context

SetFieldProperties -FieldXML $FieldXML -Context $Context

Write-Verbose('La columna {0} se creó correctamente' -f $FieldXML.DisplayName)

} else {

Write-Warning('La columna {0} ya existe, actualizando' -f $FieldQuery.Title)

UpdateFieldInfo -Field $FieldQuery -Context $Context -UpdateInfo $FieldXML

Write-Verbose('La columna {0} se actualizó correctamente' -f $FieldQuery.Title)

}

}

<#

Crea el campo especificado en el XML

#>

Function CreateField {

param ($FieldXML, $Context)

# Si es de taxonomía o de Media, llama a las funciónes específicas

if($FieldXML.FieldType -eq "TaxonomyFieldType") {

CreateTaxonomyField -FieldXML $FieldXML

} elseif($FieldXML.FieldType -eq "MediaFieldType"){

CreateMediaField -FieldXML $FieldXML -Context $Context

} else {

Add-SPOField -DisplayName $FieldXML.DisplayName -InternalName $FieldXML.Name -Type $FieldXML.FieldType -Id $FieldXML.Id -Group $FieldXML.Group

}

}

<#

Crea el campo de taxonomía cuya información se recibe desde el XML

#>

Function CreateTaxonomyField {

param ($FieldXML)

# Sacamos el TermSetPath en el formato que espera el método Add-SPOTaxonomyField 'NombreAlmacen|NombreGrupo'

$TaxonomyInfo = $FieldXML.SelectSingleNode("TaxonomyFieldType")

$TermSetName = $TaxonomyInfo.TermSetName

$TermStoreName = $TaxonomyInfo.TermStoreGroupName

$TermSetPath = "{0}|{1}" -f $TermStoreName, $TermSetName

Add-SPOTaxonomyField -DisplayName $FieldXML.DisplayName -InternalName $FieldXML.Name -Group $FieldXML.Group -Id $FieldXML.Id -TermSetPath $TermSetPath

}

<#

Crea el campo de Media. Es necesario utilizar el CSOM puro ya que el PnP no soporta el MediaFieldType

#>

Function CreateMediaField {

param ($FieldXML, $Context)

$Fields = $Context.Site.RootWeb.Fields

Execute-SPOQuery

# Añadimos los Assemblies

[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SharePoint.Client") | Out-Null

[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SharePoint.Client.Runtime") | Out-Null

# Creamos el XML

$ReadyXML = "<Field DisplayName='{0}' Name='{1}' ID='{2}' Group='{3}' Type='MediaFieldType' StaticName='{4}'/>" -f $FieldXML.DisplayName, $FieldXML.Name, $FieldXML.Id, $FieldXML.Group, $FieldXML.StaticName

$Field = $Fields.AddFieldAsXml($ReadyXML, $false, [Microsoft.SharePoint.Client.AddFieldOptions]::AddFieldToDefaultView)

$Field.Update();

Execute-SPOQuery

}

<#

Añande las propiedades de la columna especificada para aquellas que no permite establecer al crearlas

#>

Function UpdateFieldInfo {

param ($Field, $Context, $UpdateInfo)

$Field.Title = $UpdateInfo.DisplayName

$Field.Group = $UpdateInfo.Group

$Field.StaticName = $UpdateInfo.StaticName

$Field.Hidden = [System.Convert]::ToBoolean($UpdateInfo.Hidden)

$Field.Required = [System.Convert]::ToBoolean($UpdateInfo.Required)

$Field.SetShowInDisplayForm([System.Convert]::ToBoolean($UpdateInfo.ShowInDisplayForm))

$Field.SetShowInEditForm([System.Convert]::ToBoolean($UpdateInfo.ShowInEditForm))

$Field.SetShowInNewForm([System.Convert]::ToBoolean($UpdateInfo.ShowInNewForm))

$Field.Update()

Execute-SPOQuery

}

<#

Estable las propiedades del campo recién creado

#>

Function SetFieldProperties {

param ($FieldXML, $Context)

$Field = Get-SPOField -Identity $FieldXML.Name -ErrorAction SilentlyContinue

Execute-SPOQuery

UpdateFieldInfo -Field $Field -Context $Context -UpdateInfo $FieldXML

}

Export-ModuleMember -Function CreateOrUpdateFieldFromXML

A Continuación nos definimos las funciones para trabajar con Tipos de contenido:

<#

Esta función creará los ContentTypes que indiquen los XML que existan en la carpeta especificada

#>

Function CreateContentTypes {

param ($RootFolder, $UrlSPOnline)

Connect-SPOnline -Url $UrlSPOnline –Credentials premise -ErrorAction Stop

$Context = Get-SPOContext

# Obtenemos todos los ficheros .xml que comiencen por ContentTypes

Get-ChildItem ('{0}\' -f $RootFolder) -Filter ContentTypes\*.xml | `

ForEach-Object{

# Por cada uno de ellos, recorremos todos los nodos Add y añadimos o actualizamos los tipos de contenido especificados

[xml]$XML = Get-Content $\_

$ContentTypeDefinition = $XML.SelectNodes("/Site/ContentTypes/Add")

foreach( $ContentType in $ContentTypeDefinition) {

$ContentTypeQuery = Get-SPOContentType -Identity $ContentType.Name

Execute-SPOQuery

if($ContentTypeQuery.Id -eq $null) {

Add-SPOContentType -Name $ContentType.Name -ContentTypeId $ContentType.ID -Group $ContentType.Group -Description $ContentType.Description

Write-Verbose("El tipo de contenido {0} se creó correctamente, añadiendo columnas" -f $ContentType.Name)

} else {

Write-Verbose("El tipo de contenido {0} ya existe, actualizando" -f $ContentType.Name);

UpdateContentTypes -Context $Context -ContentTypeXML $ContentType

}

# Exista o no, se le añaden las columnas de sitio

Add-FieldsToContentType -ContentTypeXML $ContentType

}

}

}

<#

Actualiza las propiedades de los tipos de contenido susceptibles de ser actualizadas

#>

Function UpdateContentTypes {

param ($Context, $ContentTypeXML)

$ContentType = Get-SPOContentType -Identity $ContentTypeXML.Id

Execute-SPOQuery

# Obtenemos y actualizamos el Name para utilizar el DisplayName

$DisplayName = $ContentTypeXML.SelectSingleNode('//DisplayName').Value

$ContentType.Name = $DisplayName

$ContentType.Description = $ContentTypeXML.Description

$ContentType.Group = $ContentTypeXML.Group

$ContentType.Update($true)

Execute-SPOQuery

}

<#

Añade los campos específicados en el XML al content type también especificado en el XML

#>

Function Add-FieldsToContentType {

param ($ContentTypeXML)

$CamposAAgregar = $ContentTypeXML.SelectNodes("//Fields/Add")

foreach($Campo in $CamposAAgregar) {

Add-SPOFieldToContentType -Field $Campo.InternalName -ContentType $ContentTypeXML.Name

Write-Verbose("Columna {0} añadida con éxito" -f $Campo.InternalName)

}

}

Export-ModuleMember -Function CreateContentTypes

Ahora nos definiremos un método parar trabajar con Listas y Bibliotecas de Documentos, este módulo es el siguiente:

<#

Esta función crea un Lista o una biblioteca de documentos dependiendo de la información contenida en el manifest.xml

#>

Function CreateListOrDocLib {

param($Context, $RootFolder, $UrlSPOnline)

Get-ChildItem -Path $RootFolder -Filter manifest.xml -Recurse | `

Where-Object {$\_.DirectoryName -like '\*LIST-\*' -OR $\_.DirectoryName -like '\*DOCLIB-\*' -and $\_.DirectoryName -notlike '\*PAGE-\*' -and $\_.DirectoryName -notlike '\*ITEM\*'} | `

ForEach-Object {

Connect-SPOnline -Url $UrlSPOnline –Credentials premise -ErrorAction Stop

$Context = Get-SPOContext

[xml]$ListManifest = Get-Content -Path ('{0}/manifest.xml' -f $\_.Directory)

$Lists = $ListManifest.SelectNodes('/List[not(@EnableRating)]')

$DocLibs = $ListManifest.SelectNodes('/List[@EnableRating]')

$Web = GetListWeb -Context $Context -RootFile $\_

foreach($List in $Lists) {

CreateList -List $List -Web $Web

UpdateListOrDocLibInfo -ListOrDocLib $List -Context $Context -Web $Web

AddContentTypeToList -ListManifest $List -WebCT $Web

}

foreach($DocLib in $DocLibs) {

CreateDocLibs -Library $DocLib -Web $Web

UpdateListOrDocLibInfo -ListOrDocLib $DocLib -Context $Context -Web $Web

AddContentTypeToList -ListManifest $DocLib -WebCT $Web

}

}

}

<#

Obtiene la Web a la que está asociada la lista. Es necesario usar full CSOM ya que con el método de PnP no funciona bien

#>

Function GetListWeb {

param($Context, $RootFile)

[xml]$WebManifest = Get-Content -Path ('{0}/manifest.xml' -f $RootFile.Directory.Parent.FullName)

$SiteUrl = $WebManifest.SelectSingleNode('//RelativeUrl').'#text'

if($SiteUrl -eq '/') {

$Web = $Context.Site.RootWeb

} else {

$Web = $Context.Site.OpenWeb('/sites/sareb{0}' -f $SiteUrl)

}

$Context.Load($Web)

return $Web

}

<#

Crea la lista que se le suministra

#>

Function CreateList {

param($List, $Web)

$ListQuery = Get-SPOList -Identity $List.Name -ErrorAction SilentlyContinue -Web $Web

Execute-SPOQuery

if($ListQuery -eq $null) {

New-SPOList -Title $List.Name -Url $List.Url -Template GenericList -Web $Web

Write-Verbose('La lista {0} se creó correctamente' -f $List.Name)

} else {

Write-Verbose('La lista {0} ya existe' -f $List.Name)

}

}

<#

Crea la biblioteca que se le suminstra

#>

Function CreateDocLibs {

param($Library, $Web)

$LibraryQuery = Get-SPOList -Identity $Library.Name -ErrorAction SilentlyContinue -Web $Web

Execute-SPOQuery

if($LibraryQuery -eq $null) {

New-SPOList -Title $Library.Name -Url $Library.Url -Template DocumentLibrary -Web $Web

Write-Verbose('La biblioteca {0} se creó correctamente' -f $List.Name)

} else {

Write-Verbose('La biblioteca {0} ya existe' -f $List.Name)

}

}

<#

Actualiza la información susceptible de actualizarse utilizando el fichero XML suminsitrado

#>

Function UpdateListOrDocLibInfo {

param($ListOrDocLib, $Context, $Web)

Write-Verbose('Actualizando {0}' -f $ListOrDocLib.Name)

#Obtenemos la lista

$List = Get-SPOList -Identity $ListOrDocLib.Name -Web $Web

Execute-SPOQuery

$List.Description = $ListOrDocLib.Description

<#

Si la lista es una biblioteca de documentos, actualizamos la propiedad EnableModeration.

#>

if(!($ListOrDocLib.EnableModeration -eq $null)) {

$List.EnableModeration = [System.Convert]::ToBoolean($ListOrDocLib.EnableModeration)

}

$List.Update()

Execute-SPOQuery

}

<#

Añade el content typeo especificado a la lista especificada

#>

Function AddContentTypeToList {

param($ListManifest, $WebCT)

$ContentTypes = $ListManifest.SelectNodes('//ContentType')

Write-Verbose('Agregando tipos de contenido a {0}' -f $ListManifest.Name)

foreach($ContentType in $ContentTypes) {

Add-SPOContentTypeToList -List $ListManifest.Name -ContentType $ContentType.Name -Web $WebCT

Write-Verbose('Tipo de contenido {0} agragado con éxito' -f $ContentType.Name)

}

}

Finalmente tendremos un módulo para poder trabajar con Sitio/SubSitio:

<#

Crea sitios y subsitios dependiendo del fichero manifest que encuentre en las carpetas de cada web

#>

Function CreateSitesAndSubSites {

param ($SiteRoot, $UrlSPOnline, $ParentSiteUrl)

Get-ChildItem -Path $SiteRoot | Where-Object{$\_.BaseName -like '\*WEB-\*'} | `

ForEach-Object {

Connect-SPOnline -Url $UrlSPOnline –Credentials premise -ErrorAction Stop

$Context = Get-SPOContext

[xml]$SiteXml = Get-Content -Path ('{0}/manifest.xml' -f $\_.FullName)

$SiteProperties = $SiteXml.Site

# Hacemos un substring de la url relativa para obtener el leafUrl

$ForwardSlashIndex = $SiteProperties.RelativeUrl.LastIndexOf('/') + 1

$SiteUrl = $SiteProperties.RelativeUrl.Substring($ForwardSlashIndex)

# Distingue entre la creación de un sitio y la de un subsitio mediante el parametro ParentSiteUrl

if($ParentSiteUrl -eq $null) {

CreateSite -SiteProperties $SiteProperties -SiteUrl $SiteUrl

} else {

CreateSubSite -SiteUrl $SiteUrl -ParentSiteUrl $ParentSiteUrl -SiteProperties $SiteProperties

}

$Features = $SiteXml.SelectNodes('//Feature')

# Llamada recursiva añadiendo el parametro ParentSiteUrl para diferenciar la creacion de subsitios

CreateSitesAndSubSites -SiteRoot $\_.FullName -Context $Context -ParentSiteUrl $SiteUrl

}

}

<#

Crea un subsitio utilizando la url extraida del manifest y la url del padre que le pasemosS

#>

Function CreateSubSite {

param($SiteUrl, $ParentSiteUrl, $SiteProperties)

$ParentQuery = Get-SPOWeb -Identity $ParentSiteUrl

$WebQuery = Get-SPOWeb -Identity $SiteUrl

if($WebQuery -eq $null) {

New-SPOWeb -Title $SiteProperties.Name -Url $SiteUrl -Description $SiteProperties.Description -Locale $SiteProperties.Language -Template $SiteProperties.Template -Web $ParentQuery

} else {

Write-Warning('El sitio ya existe. Actualizando Master Pages, creando subsitio')

}

}

<#

Crea un subsitio utilizando la url extraida del manifest

#>

Function CreateSite {

param ($SiteProperties, $SiteUrl)

$WebQuery = Get-SPOWeb -Identity $SiteUrl

if($WebQuery -eq $null) {

New-SPOWeb -Title $SiteProperties.Name -Url $SiteUrl -Description $SiteProperties.Description -Locale $SiteProperties.Language -Template $SiteProperties.Template

} else {

Write-Warning('El sitio ya existe. Actualizando Master Pages, creando subsitio')

}

}

Export-ModuleMember -Function CreateSitesAndSubSites

Por último queda definirnos el Script que lanzaremos para que llame a todas las funciones establecidas, este script puede ser el siguiente:

# Cambiar a 'SilentlyContinue' si no queremos información sobre el progreso del Script

$VerbosePreference = 'Continue'

# Cargamos los módulos necesarios

Import-Module ("{0}/Modules/SiteColumn.psm1" -f $PSScriptRoot) -Force

Import-Module ("{0}/Modules/ContentTypes.psm1" -f $PSScriptRoot) -Force

Import-Module ("{0}/Modules/Sites.psm1" -f $PSScriptRoot) -Force

Import-Module ("{0}/Modules/ListsDocLibs.psm1" -f $PSScriptRoot) -Force

# -- Obtencion de Conexion y Contexto, si no se conecta, por el motivo que sea, paramos el Script

$UrlSPOnline = "https://nombredesvuestositio.sharepoint.com/sites/nombredelsitio/"

Connect-SPOnline -Url $UrlSPOnline –Credentials premise -ErrorAction Stop

$Context = Get-SPOContext

# -- Creación de las columnas de sitio a partir del XML especificado

[xml]$Xml = Get-Content -Path ('{0}/SiteColumns.xml' -f $PSScriptRoot)

$Campos = $Xml.SelectNodes("//Add")

foreach($Campo in $Campos) {

CreateOrUpdateFieldFromXML -FieldXML $Campo -Context $Context

}

Execute-SPOQuery

# -- Creación de los ContentTypes

CreateContentTypes -RootFolder $PSScriptRoot -UrlSPOnline $UrlSPOnline

# -- Creacion de subsitios

CreateSitesAndSubSites -SiteRoot $PSScriptRoot -UrlSPOnline $UrlSPOnline

# -- Creación de listas y bibliotecas

CreateListOrDocLib -RootFolder $PSScriptRoot -UrlSPOnline $UrlSPOnline