# Contribution guidance

All PnP repositories are following up on the standard PnP process on getting started and contribute.

See following PnP wiki page from the main repository for additional details.

- For getting started guidance, see [Setting up your environment](https://github.com/OfficeDev/PnP/wiki/Setting-up-your-environment).

\*Notice that you'll need to update the URLs based on used repository. All community contributions are also more than welcome.

Please see following page for additional insights on the model.

- For contributing to PnP, see [Contributing to Office 365 developer patterns and practices](https://github.com/OfficeDev/PnP/wiki/contributing-to-Office-365-developer-patterns-and-practices)

---

## Building the source code ##

Once you have downloaded the code, in the folder with the PnP PowerShell source code, open the solution file SharePointPnP.PowerShell.sln.

If you have set up the projects and you are ready to build the source code, make sure to build the SharePointPnP.PowerShellModuleFilesGenerator project first. This project will be executed after every build and it will generate the required PSD1 and XML files with cmdlet documentation in them.

When you build the solution a postbuild script will copy the required files to a folder in your users folder called

\*C:\Users\\\<YourUserName\>\Documents\WindowsPowerShell\Modules\SharePointPnPPowerShell\<Platform\>\*. During build also the help and document files will be generated. If you have a session of PowerShell open in which you have used the PnP Cmdlets, make sure to close this PowerShell session first before you build. You will receive a build error otherwise because it tries to overwrite files that are in use.

To debug the cmdlets: launch PowerShell and attach Visual Studio to the powershell.exe process. In case you want to debug methods in PnP Sites Core, make sure that you open the PnP Sites Core project instead, and then attach Visual Studio to the powershell.exe. In case you see strange debug behavior, like it wants to debug PSReadLine.ps1, uninstall the PowerShell extension from Visual Studio.

## Keeping your fork up to date

Before starting on any new submissions, please ensure your fork is up to date with the upstream repository. This avoids frustration and challenges for us to valdate and test your submission. Steps on how to easily keep your fork up to date can be found [on this wiki page](https://github.com/pnp/PnP-PowerShell/wiki/Update-your-fork-with-the-latest-code).

## Code contributions

In order to succesfully compile the PnP PowerShell solution you will \_also\_ have to download \*and build in Visual Studio\* the [PnP-Sites-Core](https://github.com/OfficeDev/PnP-Sites-Core) repository and make the dev branch available. The PowerShell solution depends on it. In order to succesfully

compile it, make sure that PnP-Sites-Core is located at the same level as PnP-PowerShell and you open the solution file OfficeDevPnP.Core.sln located in the Core subfolder of the sourcecode.

So:

```

c:\[YOUR REPO FOLDER]\PnP-Sites-Core

c:\[YOUR REPO FOLDER]\PnP-PowerShell

```

The reason for this is that the PnP-PowerShell library will have references to the release and debug builds of the PnP-Sites-Core library.

A few notes:

### Every new cmdlet should provide help and examples

As documentation is autogenerated by building the solution, make sure that you include both help and examples, alike

```csharp

[Cmdlet(VerbsCommon.Get, "SPOStoredCredential")]

[CmdletHelp("Returns a stored credential from the Windows Credential Manager", Category = "Base Cmdlets")]

[CmdletExample(Code = "PS:> Get-SPOnlineStoredCredential -Name O365",

Remarks = "Returns the credential associated with the specified identifier",

SortOrder = 1)]

public class GetStoredCredential : PSCmdlet

{

}

```

### Most cmdlets will extend PnPCmdlet or PnPWebCmdlet which provides a few helper objects for you to use, like SelectedWeb and ClientContext

As most cmdlets are 'web sensitive' (e.g. you can specify a -Web parameter to point to a different subweb), make sure that you use the correct ClientContext. When a user specifies the -Web parameter

in a cmdlet that extens PnPWebCmdlet, the cmdlet will switch it's internal context to that web, reusing credentials. It is important to use the right context, and the easiest way to do that is to use

```csharp

var context = ClientContext

```

alternatively

```csharp

var context = SelectedWeb.Context;

```

### Cmdlets will have to work both on-premises and in the cloud

You can use preprocessor variables ("ONPREMISES" or "SP2013" and "SP2016") to build different cmdlets for the different targets. In cases where it is not possible to provide functionality for either the

cloud or on-premises, make sure to remove the full cmdlet from the compiled solution by having #IF(!SP2013) or #IF(SP2013) as the \_first line of the cmdlet, before using statements.

See the following example

```csharp

#if !ONPREMISES

using Microsoft.SharePoint.Client;

public class MyCmdlet : SPOWebCmdlet

{

// cmdlet code omitted

}

#endif

```

If only parts of a cmdlet require different behavior based upon the different version of the SDK, you are recommended to use the #ONPREMISES or other available preprocessor variable throughout your code to exclude or include certain code.

### Cmdlets will have to use common verbs

The verb of a cmdlet (get-, add-, etc.) should follow acceptable cmdlet standards and should be part of one of the built in verbs classes (VerbsCommon, VerbsData, etc.):

## Documentation contributions

The PowerShell documentation is located on [docs.microsoft.com](https://docs.microsoft.com/en-us/powershell/sharepoint/sharepoint-pnp/sharepoint-pnp-cmdlets?view=sharepoint-ps)

This documentation is automatically generated based upon statements/attributes in the source code of a cmdlet. If you want to modify documentation or add an example of a cmdlet, navigate to the

corresponding class where the cmdlet is being implemented and add the comments there. An example can for instance be found in

https://github.com/sharepoint/PnP-PowerShell/blob/dev/Commands/Fields/AddField.cs

Notice the [CmdletHelp("")] and [CmdletExample()] class attributes that describe the cmdlet. A description for a parameter is located in the [Parameter(HelpMessage = "") attribute.