# How to contribute

We welcome contributions to the NuGet Gallery! However, we do have a few suggestions for making sure we can integrate your code in the cleanest possible way.

## Choosing something to work on

We love contributions, but please remember that we have to maintain all the code we accept. Sometimes we'll decide that a feature or other change is not in our primary goals and will decline the request. This doesn't mean we don't like it! We just don't think it fits well in our plan for the main gallery. Feel free to disagree with us and start a conversation!

Similarly, if you submit a giant Pull Request modifying a bunch of different files without any warning or discussion in advance, we will probably decline it. If you want to take on a big task, give us a heads-up first by commenting on the relevant issue (if there isn't one, create one first). We'll let you know how we feel about it so you know before you write your code.

Finally, don't be afraid to fork! If you need a feature for your own deployment of the Gallery, don't hesitate to maintain a fork.

### Up For Grabs

A lot of the issues that arise in our tracker are things we'd like to do and would be happy to maintain, but just don't have the resources to devote to building. We place those in the [Up For Grabs](https://github.com/NuGet/NuGetGallery/issues?milestone=13&page=1&state=open) milestone in our issue tracker. Feel free to grab any of those issues and start working on them (though again, a quick comment indicating you're going to work on it is always appreciated).

## Getting Started

1. Make sure there is an issue in the [Issue Tracker](https://github.com/NuGet/NuGetGallery/issues) for the feature or bug you want to fix. If not, create one!

\* \*\*Pull Requests without an associated issue will not be accepted!\*\*

1. Create a fork in GitHub

1. Create a branch off of the \*\*master\*\* branch. Name it something which easily links it back to the issue. For example: "Bug-1234".

1. Make your changes

1. Send a Pull Request from your branch to the \*\*master\*\* branch.

If you mess up the branching or get confused, that's OK, we'd rather have your contribution than have you waste a lot of time figuring out our branches. However, using the right branches will help us immensely and will make it much easier for us to quickly accept your contribution!

## DOs and DON'Ts

\* \*\*DO\*\* follow the code style of the rest of the gallery. When you can't figure out what the style for a particular aspect is, please bring that up in your PR and we'd be happy to advise you.

\* \*\*DON'T\*\* surprise us with big Pull Requests. If you have a feature you want to work on, file an Issue so we can discuss it.

\* \*\*DO\*\* grab things from the "Up For Grabs" milestone! Some of those things are small things we just don't have time to get to, but we'd be happy to take contributions. Think of it as a great way to get your feet wet :)

\* \*\*DO\*\* remember that we have to dedicate resources to integrate and test your code, so it may take a while for us to integrate it. Feel free to ping the PR and nag us to take a look :)

\* \*\*DO\*\* include Unit Tests in your change. We're also building out our Functional Tests, so the more of those you provide, the more excited we'll be to accept your change :).

\* \*\*DO\*\* ensure your code is free of compilation warnings, FxCop errors/warnings and Unit or Functional Test failures.

\* \*\*DO\*\* remember that we will be maintaining the code you contribute, so there may be cases where we have to decline a feature because it doesn't make much sense in the main code. You are more than welcome to maintain a custom fork of the code for your use cases.

## Specific code style notes

\*\*Note\*\*: Even we don't always follow these in older code, but new code should follow them.

1. We use Allman style braces (the VS default)

1. We avoid "this." unless absolutely necessary

1. We always specify the visiblity, even if it's the default (i.e. "private string \_foo" not "string \_foo")

1. We avoid unnecessary abbreviations. Use 'Service' not 'Svc', etc.

1. We use "\_camelCase" private members and use "readonly" where possible

1. Our tests use the [Arrange/Act/Assert](http://c2.com/cgi/wiki?ArrangeActAssert) pattern

1. Namespace Imports should be specified at the top of the file, OUTSIDE of "namespace" declarations and should be sorted alphabetically, with 'System.' namespaces at the top (there is a VS setting to do this):

1. Namespaces should, in general, match folder structure.

1. We use 4 spaces of indentation (in general, preserve indentation used by the file and always use spaces).

### Example File:

```C#

using System.Aardvarks;

using System.IO;

using System.Zebras;

using Aardvarks.CoolStuff;

using Zebra.Crossing;

namespace NuGetGallery.FolderName.SubFolderName

{

public class MyClass

{

private readonly ISomethingService \_something;

public MyClass(ISomethingService something)

{

\_something = something;

}

}

}

```