Publishing QGIS Plugins in a Custom Repository

Learn how to prepare you plugins for hosting in a custom repository and share them with other QGIS users

Yaroslav Vasyunin Follow
Oct 8 · 6 min read

This is my translation of a post published on habr.com (authors: Sergei Seminozhenko and Vasily Lobanov).

Most of the available QGIS plugins are hosted in the official repository. Sometimes, it becomes necessary to create a custom repo, for example, to distribute a plugin inside your organization. This post explains the essential requirements for plugins, shows you a few simple examples, and, finally, demonstrates how to make your own plugin repository.

The information given here is relevant to QGIS 3.14 'Pi', but it should also work for other versions. Note that publishing requirements for plugins in the official QGIS repository are more strict than in a custom one. Therefore, if you are going to publish a plugin in the official repo, I recommend you to check the following links:

- Learn how to publish your plugins (QGIS plugins web portal)
- Releasing your plugin (PyQGIS Developer Cookbook)

Plugin Manager

QGIS has a tool called **Plugin Manager (PM)**. It allows you to search for plugins, install, update and delete them, connect third-party repositories, and so on. For each plugin,

PM provides an information window with fields showing plugin's metadata. There are three scenarios of how PM retrieves plugin metadata:

1. **A plugin is in a repository, but not installed on your computer.** All the metadata is extracted from the `plugins.xml` file.
2. **A plugin is installed on your computer, and there is a connection with a repository.** Some of the metadata is still extracted from the plugin description in `plugins.xml` and some from the `metadata.txt` file.
3. **A plugin is installed on your computer, and there is no connection to any repository.** All metadata about the plugin is extracted from the `metadata.txt` file.

The following table describes, for every scenario, where plugin metadata comes from (see also comments below it):

Search this file…

Field in Plugin Manager	Before installation from a repository	A plugin is installed and connected to
Name	plugins.xml (attribute 'name' of 'pyqgis_plugin' tag)	plugins.xml (attribute 'name' of 'pyqgis
Description	plugins.xml	metadata.txt
About	plugins.xml	metadata.txt
Installed version	—	metadata.txt ('version' parameter)
Available version	plugins.xml (attribute 'version' of 'pyqgis_plugin' tag)	plugins.xml (attribute 'version' of 'pyqg
Download url	plugins.xml	plugins.xml
Changelog	—	metadata.txt
Tags	plugins.xml	metadata.txt
Author	plugins.xml	plugins.xml
Email	—	metadata.txt
Homepage	plugins.xml	plugins.xml
Bug tracker	plugins.xml	plugins.xml
Code repository	plugins.xml	plugins.xml
QgisMinimumVersion	plugins.xml	metadata.txt
QgisMaximumVersion	plugins.xml	metadata.txt
Experimental	plugins.xml	metadata.txt
Deprecated	plugins.xml	metadata.txt
Icon	plugins.xml	metadata.txt
Category	plugins.xml	plugins.xml

qgis-plugin-manager.csv hosted with ♡ by GitHub view raw

Where does QGIS Plugin Manager take metadata about plugins?

- The information from `metadata.txt` and `plugins.xml` doesn't affect the `Name`, `Icon`, and placement of a plugin in QGIS menu. You provide this metadata in the `initGui()` method of your plugin's class.
- If `Installed version` is smaller than `Available version`, QGIS will propose you to update the plugin. If `Installed version` is larger than `Available version`, QGIS shows a warning: "Installed version of this plugin is higher than any version found in repository." In both cases, you can update the plugin to the version available in the repo.
- Before you upload a plugin to the repository, make sure that the plugin versions in `metadata.txt` and `plugins.xml` match. Otherwise, PM will continuously suggest to update it.

- `Available version` parameter is taken from `plugins.xml`: `version` attribute of the `pyqgis_plugin` tag(not from the `version` tag).
- If `metadata.txt` and `plugins.xml` don't specify `Experimental` and `Deprecated` parameters, a plugin will be considered non-experimental and non-outdated by default.
- The version of your QGIS installation must be between `QgisMinimumVersion` and `QgisMaximumVersion` provided in `metadata.txt` and `plugins.xml`.
- If `QgisMinimumVersion` isn't specified, PM sets it by default to `0`. If `QgisMaximumVersion` is absent, then PM sets it to `+0.99`.

Structuring Plugin Files

Each plugin in a repository should be packed into a single zip archive containing a root folder with plugin files, including `metadata.txt`. By a root folder name, PM is able to match a locally installed plugin with the plugin in a repository. I explain this behavior in the end of this post.

There are two required files in the root directory of a QGIS plugin:

- `__init__.py` is the starting point of the plugin. The code should contain the `classFactory()` method.
- `metadata.txt` contains general plugin's metadata.

You can explore a "skeleton" of the simplest QGIS plugin in this GitHub repo, or learn about structuring QGIS plugins from the PyQGIS Developer Cookbook.

metadata.txt

As already noted, each plugin's folder must contain `metadata.txt` that includes general information about the plugin. This metadata is used to be displayed in the QGIS PM, for searching in a repository, and so on.

The structure of the metadata file must be as follows:

```
[general]
Parameter1_Name: Parameter1_Value
Parameter2_Name: Parameter2_Value
...
ParameterN_Name: ParameterN_Value
```

or

```
[general]
Parameter1_Name = Parameter1_Value
Parameter2_Name = Parameter2_Value
...
ParameterN_Name = ParameterN_Value
```

Note that the `[general]` keyword at the beginning is required.

Use `;` or `=` as a separator between the parameter name and its value. You can even mix two types of delimiters in one metadata file.

Spaces before and after the separator don't affect data reading from `metadata.txt` — they can be in any quantity or absent at all.

The following are mandatory parameters, without which a plugin's operation may be broken:

- `name` — plugin's name
- `version` — plugin's version
- `qgisMinimumVersion` — minimum QGIS version suitable for the plugin

You can find information about other parameters in the PyQGIS Developer Cookbook.

plugins.xml

Each repository must contain a `plugins.xml` file that provides general information for PM about all the plugins available in the repository. In fact, the repository description file can have any name.

The structure of `plugins.xml` should be as follows:

```
<plugins>
  <pyqgis_plugin name="Plugin1_Name" version="Plugin1_Version">
    <Plugin1_Parameter1_Name>Plugin1_Parameter1_Value</Plugin1_Parameter1_Name>
    <Plugin1_Parameter2_Name>Plugin1_Parameter2_Value</Plugin1_Parameter2_Name>
    ....
    <Plugin1_ParameterN_Name>Plugin1_ParameterN_Value</Plugin1_ParameterN_Name>
  </pyqgis_plugin>
  ...
  <pyqgis_plugin name="PluginN_Name" version="PluginN_Version">
    <PluginN_Parameter1_Name>PluginN_Parameter1_Value</PluginN_Parameter1_Name>
    <PluginN_Parameter2_Name>PluginN_Parameter2_Value</PluginN_Parameter2_Name>
    ....
    <PluginN_ParameterN_Name>PluginN_ParameterN_Value</PluginN_ParameterN_Name>
  </pyqgis_plugin>
</plugins>
```

plugins.xml hosted with ♡ by GitHub view raw

Here are the mandatory parameters without which your repository may be broken:

- `name` of a plugin indicated as an attribute of the `pyqgis_plugin` tag
- `version` of a plugin shown as an attribute of the `pyqgis_plugin` tag
- `qgis_minimum_version` is the minimum version of QGIS suitable for the plugin
- `download_url` — location of zip archive with your plugin in the web

It's a good idea to look at the example of the `plugins.xml` file from the official QGIS repository: `https://plugins.qgis.org/plugins/plugins.xml?qgis=3.14`. The numbers at the end of the URL are replaced with the version of the QGIS you use. Save this page from your browser to your computer, and you get the desired XML file.

Binding a Local Plugin to a Plugin from Repository

How does PM understands that a plugin installed on your computer and a plugin in a connected repository are the same, for example, when checking whether a new version of the plugin is available in a repo?

A local plugin and a plugin in the repo are considered the same if the name of the local plugin root folder matches:

- characters in the value of the `file_name` tag until to the first dot:

  `<file_name>`**`PluginName`**`.2.1.0.zip</file_name>`

- characters from the `download_url` tag until the first dot of the last path element, if the `file_name` tag is missing:

  `<download_url>YourRepositoryURL/`**`PluginName`**`.2.1.0.zip</download_url>`

If the name of the root folder and the extracted part from `file_name` or `download_url` do not match, then the plugins are considered different. Examples of Well-designed Plugins

> I recommend you **to add the `file_name` tag to the `plugins.xml` file and specify not the name of your zip archive, but the name of your plugin's root folder**, to which you can optionally add a version or other information separated by dots. In this case, you can name your plugin's zip archive whatever you like, e.g., separating the plugin name and its version with a hyphen, as accepted in the official repository, where the `file_name` tag is used to link local and remote plugins.

A Simple Example

File Structure

```
FolderName.SomeInfo.zip
--FolderName
----__init__.py
----metadata.txt
```

metadata.txt

```
[general]
name: PluginName
version: 2.1.0
qgisMinimumVersion: 3.0
```

plugins.xml

In this variant, we link a plugin installed locally and a remote plugin in the repo through the name of the root folder and the `download_url` tag. A part of the zip archive's name until the first dot and the name of the plugin root folder must match exactly.

```
<plugins>
  <pyqgis_plugin name="PluginName" version="2.1.0">
    <qgis_minimum_version>3.0.0</qgis_minimum_version>
      <download_url>YourRepositoryURL/FolderName.SomeInfo.zip</download_url>
  </pyqgis_plugin>
```

```
</plugins>
```

plugins.xml hosted with ♡ by GitHub view raw

In the following variant, a local plugin and a remote plugin are linked by the root folder name and the `file_name` tag. In this case, the part of the zip archive name until the first dot and the root folder name may differ.

```
<plugins>
  <pyqgis_plugin name="PluginName" version="2.1.0">
    <qgis_minimum_version>3.0.0</qgis_minimum_version>
      <file_name>FolderName.SomeInfo</file_name>
      <download_url>YourRepositoryURL/ArchiveName.SomeInfo.zip</download_url>
  </pyqgis_plugin>
</plugins>
```

plugins.xml hosted with ♡ by GitHub view raw

Creating a Repository

A QGIS plugin repository should be available as a static web site and consist of a plugins.xml file that provides general information to QGIS PM about all plugins available in the repo and corresponding zip archives containing those plugins.

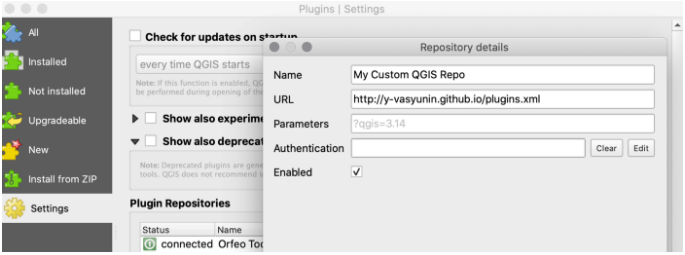
An essential requirement for QGIS to recognize a repository is **direct file access**. This can be done using web services like GitHub or Bitbucket. Cloud storages like Google Drive or Dropbox won't work for these purposes because they don't provide such direct links to resources.

For instructions on hosting a static website, see GitHub Pages and Publishing a Website on Bitbucket Cloud.

Typically, you just need to rename your repository to match the pattern: `username.github.io` (for GitHub) or `username.bitbucket.io` (for Bitbucket).

After the necessary settings, the link to your QGIS repository will look like `username.github.io/plugins.xml` (for GitHub) or `username.bitbucket.io/plugins.xml` (for Bitbucket).

This link must be specified in the URL field in the QGIS PM when adding a new repository. If the connection to the repository is successful, the status of the repository will take the value "connected", and the plugins described in the `plugins.xml` file will appear in the list of available (Not installed) plugins in QGIS PM.



Now you are ready to distribute your plugins!

Qgis Python Repositories