thoughtbot

How to Git with Unity

Rick Reilly – June 21, 2017 UPDATED ON April 12, 2019

UNITY, GIT, VR

Unity is awesome. Git is awesome. Wouldn't it be nice if they got along?

The Problem

By default, the Unity editor does not work well with Git. To name a few problems:

- **Noise**: The editor manages hundreds of temporary files. Merely opening or closing the editor will create, delete, or modify some of these files. Additionally, since it's possible to develop on and build for multiple platforms, there are many more extraneous files than you might expect to find on, say, a Rails project.
- **Broken object references**: The editor keeps track of objects with randomly generated GUIDs. If these GUIDs are not committed to Git properly (i.e., via `.meta` files), the project may break when you, for example, switch branches. The more you rely on the editor, the more common and potentially catastrophic these errors are.
- **Unresolvable merge conflicts**: Depending on your settings, the editor will save some or all of your files in binary format. If you and a team member happen to edit the same file independently – a common scenario if you use the Unity editor heavily – you will not be able to resolve the merge conflict with Git, and will have to use special tools.
- **Large files**: A typical Unity project requires a number of large binary assets for 3D models, sounds, images, fonts, etc., which can significantly slow down your Git workflow and waste storage space.

thoughtbot

The solution is straightforward:

1. Add Unity-specific .gitignore settings
2. Configure Unity for version control
3. Use Git Large File Storage

1. Add Unity-specific .gitignore Settings

We recommend GitHub's Unity .gitignore template.

In addition, depending on the platforms you intend to use for development, you should gitignore common files for macOS and/or Windows.

2. Configure Unity For Version Control

With your project open in the Unity editor:

- Open the editor settings window.
  - `Edit > Project Settings > Editor`
- Make `.meta` files visible to avoid broken object references.
  - `Version Control / Mode: "Visible Meta Files"`
- Use plain text serialization to avoid unresolvable merge conflicts.
  - `Asset Serialization / Mode: "Force Text"`
- Save your changes.
  - `File > Save Project`

This will affect the following lines in your editor settings file:

thoughtbot

- `m_SerializationMode: 2`

- If you're curious, you can read more about Unity's YAML scene format here.

3. Use Git Large File Storage

Git Large File Storage (LFS) uses Git attributes to track large files with Git, while keeping them out of your actual repository. Note that this will only work if you use GitHub or a server that supports the Git LFS API.

To set it up, download and install the Git LFS command line extension as documented on the Git LFS site.

You can manually track the file types that you'd like Git LFS to manage, as described in the Git LFS docs. However, given the numerous file types that Unity supports, you are likely to miss a few.

Instead, feel free to use this sample `.gitattributes` file, which comprehensively accounts for all the file types that Unity currently supports (either natively or via conversion):

```
# 3D models
*.3dm filter=lfs diff=lfs merge=lfs -text
*.3ds filter=lfs diff=lfs merge=lfs -text
*.blend filter=lfs diff=lfs merge=lfs -text
*.c4d filter=lfs diff=lfs merge=lfs -text
*.collada filter=lfs diff=lfs merge=lfs -text
*.dae filter=lfs diff=lfs merge=lfs -text
*.dxf filter=lfs diff=lfs merge=lfs -text
*.fbx filter=lfs diff=lfs merge=lfs -text
*.jas filter=lfs diff=lfs merge=lfs -text
*.lws filter=lfs diff=lfs merge=lfs -text
*.lxo filter=lfs diff=lfs merge=lfs -text
*.ma filter=lfs diff=lfs merge=lfs -text
*.max filter=lfs diff=lfs merge=lfs -text
```

thoughtbot

```
*.skp filter=lfs diff=lfs merge=lfs -text
*.stl filter=lfs diff=lfs merge=lfs -text
*.ztl filter=lfs diff=lfs merge=lfs -text
# Audio
*.aif filter=lfs diff=lfs merge=lfs -text
*.aiff filter=lfs diff=lfs merge=lfs -text
*.it filter=lfs diff=lfs merge=lfs -text
*.mod filter=lfs diff=lfs merge=lfs -text
*.mp3 filter=lfs diff=lfs merge=lfs -text
*.ogg filter=lfs diff=lfs merge=lfs -text
*.s3m filter=lfs diff=lfs merge=lfs -text
*.wav filter=lfs diff=lfs merge=lfs -text
*.xm filter=lfs diff=lfs merge=lfs -text
# Fonts
*.otf filter=lfs diff=lfs merge=lfs -text
*.ttf filter=lfs diff=lfs merge=lfs -text
# Images
*.bmp filter=lfs diff=lfs merge=lfs -text
*.exr filter=lfs diff=lfs merge=lfs -text
*.gif filter=lfs diff=lfs merge=lfs -text
*.hdr filter=lfs diff=lfs merge=lfs -text
*.iff filter=lfs diff=lfs merge=lfs -text
*.jpeg filter=lfs diff=lfs merge=lfs -text
*.jpg filter=lfs diff=lfs merge=lfs -text
*.pict filter=lfs diff=lfs merge=lfs -text
*.png filter=lfs diff=lfs merge=lfs -text
*.psd filter=lfs diff=lfs merge=lfs -text
*.tga filter=lfs diff=lfs merge=lfs -text
*.tif filter=lfs diff=lfs merge=lfs -text
*.tiff filter=lfs diff=lfs merge=lfs -text
```

A Bonus For GitHub Users: Automatically Collapse Generated File Diffs

If you use GitHub to review diffs (ex., as part of a pull request workflow), you'll notice that changes in Unity-generated YAML files are usually not actionable. You can reduce the clutter they introduce, while preserving the ability to review them as needed, by automatically collapsing the diffs on GitHub.

To do so, just append this to your `.gitattributes` file:

thoughtbot

```
*.mat linguist-generated
*.meta linguist-generated
*.prefab linguist-generated
*.unity linguist-generated
```

You can read more about this feature here.

Conclusion

You should now be able to use Git to version control a Unity project as you normally would:

- Any changes detected by Git will be legitimate, not noise generated by the editor.
- You will be able to commit your changes to your repo confidently, without fearing that the project will suddenly break when a team member tries to fetch it or you switch branches.
- Your large binary files can be tracked without slowing down or cluttering your repository.
- Changes to files generated by Unity (such as Scene files) will have diffs that can be inspected normally, and merge conflicts can (at least in theory) be resolved manually. Although...

Actually, About Those Merge Conflicts...

Manually resolving merge conflicts between Unity-generated YAML files is very difficult and error-prone. If you followed the steps above and you're using Unity 5 or later, you can use the editor's Smart Merge (a.k.a, "Unity YAML Merge"). There are also various merge tools on the Unity Asset Store.

As a developer, though, I find these solutions somewhat unsatisfying. The underlying problem is not that there is a merge conflict, per se, but that a tool (the Unity editor) is

thoughtbot

directly.

Fortunately, a code-centric approach to Unity development (ex., via Zenject) can minimize these kinds of problems, while also supporting numerous software development best practices that are often lacking in Unity development.

If you enjoyed this post, you might also like:

It's time for a new branch

${VISUAL}ize the Future

Announcing gitsh

Upgrade your codebase with a Code Audit

Learn how we can help you understand the current state of your code quality, speed up delivery times, improve developer happiness, and level up your user experience

Learn more about a Code Audit

Services Case Studies

thoughtbot

Our Company
Purpose

Blog
Sponsor

Twitter
GitHub

Dribbble
Instagram

Privacy Policy