metabuddy

MetaBuddy for Unity Documentation

N.B. This is the offline documentation

This is the offline version of the documentation that comes as part of the MetaBuddy Package.

You can find the online version, which may be more up-to-date, here.

Table of Contents

Requirements

1. Unity Editor **2018.4** or newer, either **Personal or Pro** edition, running on **Windows** or **macOS**

2. A Unity Project stored in a **git repository**. Here's how to add a project to git (uses GitHub).

3. **Visible Meta Files** enabled in your Unity Project.

- This is the default in modern versions of Unity so they are *probably* already enabled. If not, here's how to enable them.

Features

- Finds meta file errors **right from within the Unity Editor**.
- Helps you **catch meta file errors at source**, before they propagate throughout your team.
- Works with **any git client** (eg. GitHub, GitLab, GitKraken, git command line).
- One-click user interface that **everyone** on your team can use.
- Context menu and help icons help you to **fix errors quickly**.
- **Adaptive user interface** works with Light (Personal) and Dark (Pro) editor skins.

Installation and Quick Start

1. Download the Asset

MetaBuddy is provided as a Unity Asset. Head over to the Unity Asset Store to download it.

2. Import MetaBuddy into your project

Unity **2020.1 or newer**

- Go to **Window > Package Manager**.
- Select **Packages: My Assets** from the dropdown list.
- Click on **MetaBuddy**, then click the **Import** button at the bottom right to import it into your project.

Unity **2019.4 or earlier**

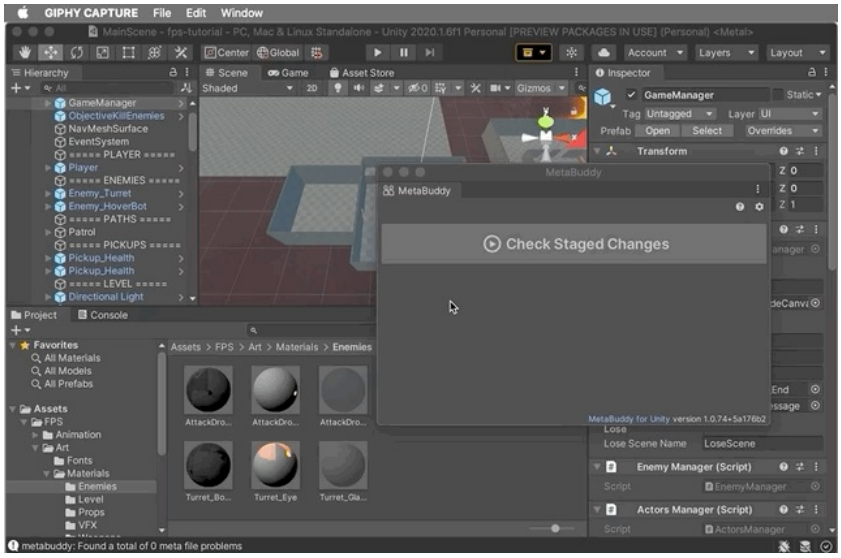
- Go to **Window > Asset Store**.
- Click the **My Assets** button at the top of the **Asset Store window**.
- Finally, click the **Import** button for the **MetaBuddy** Asset.

3. Open the MetaBuddy Window

Choose **Window->MetaBuddy** from the Unity Editor menu.

4. Dock MetaBuddy (optional)

This is down to personal preference, but we like to dock MetaBuddy as a tab beside **Inspector**. This means that it is always to hand, without getting in the way.



5. Stage your changes

Stage changes for commit using your usual git client.

MetaBuddy integrates directly with your project's git repository. This means that it works with a wide range of git clients; git command line, GitHub for Unity, GitKraken etc.

6. Check your changes

Click MetaBuddy's `Check Staged Changes` button to check your staged changes for meta file problems.

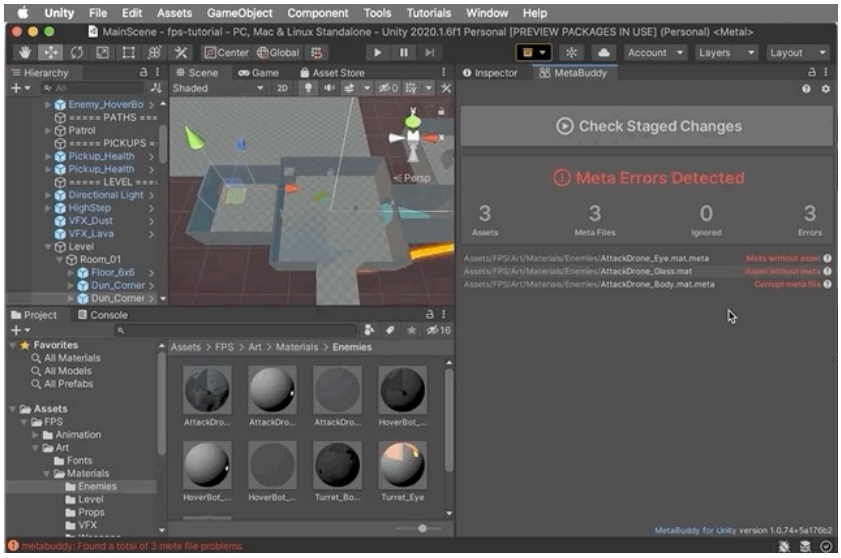
If the check came back with **No Meta Errors**, then congratulations, you are good to commit your changes using your usual git client, confident that you aren't introducing meta file problems that might impact your team.

If MetaBuddy found meta file errors in your staged changes, it will come back with **Meta Errors Detected** followed by a list of the individual errors that it found.

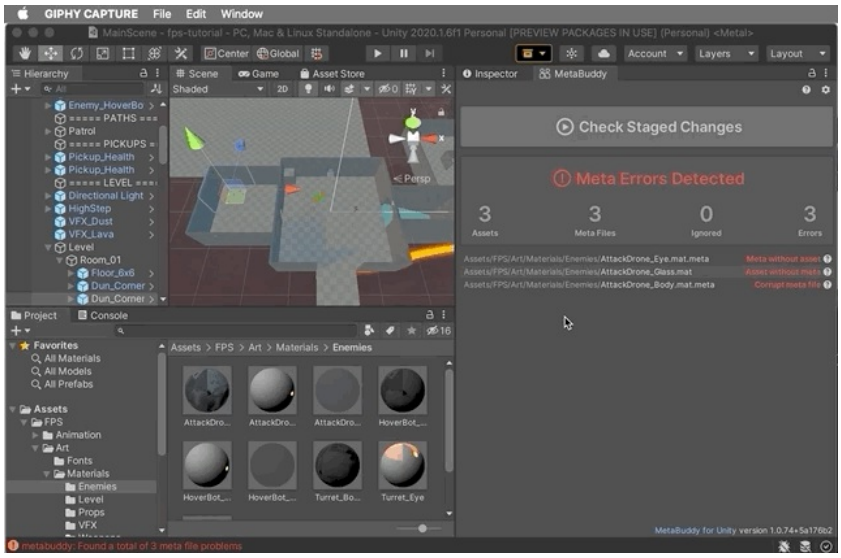
If you see an error message rather than **No Meta Errors** or **Meta Errors Detected**, check out the Troubleshooting section below for help on how to fix it.

6. Fix any errors

You can get a suggestion for how to fix each error by hovering the cursor over the red error description.



The right-click context menu for each error lets you copy the affected file's path to the clipboard, or reveal it Finder/Explorer for closer inspection.



7. Pro-tip: Click the help icon (?) to the right of each error for:

- More detail on the error.
- Common causes of the error.
- How to fix it.
- How to avoid the error in the future.

Follow the instructions to fix all of the errors, re-checking with MetaBuddy as you go until you see **No Meta Errors**.

Roadmap

We're continually working to extend and improve MetaBuddy.

Check out the backlog in GitHub to see what we're planning for future releases.

Want to vote-up a feature? Or got an idea for a new one? Don't be shy, raise an issue or drop us a line.

Errors that MetaBuddy catches

MetaBuddy catches various meta file problems that developers run into when storing Unity projects in git.

Most of these problems arise through human errors, mistakes that developers make when committing their work to git.

Asset added without adding its meta file

This error occurs when you add an asset in your Unity project to git without adding its companion `.meta` file.

For example, if you added an asset file `Assets/Textures/Explosion.png` to git, you should also add its companion meta file `Assets/Textures/Explosion.png.meta`

The meta file is important because it contains the GUID that Unity uses to keep track of the asset as well as any settings associated with the asset (for instance, which mode to compress a texture with).

How to fix this error

Usually when you see this error, the meta file for the asset already exists and you just need to stage it for addition to git with `git add <META-FILENAME>` or your git client's equivalent.

For example `git add Assets/Textures/Explosion.png.meta`

How to avoid in the future

When adding new assets to git, make sure you add both the asset **and** its meta file.

Directory added without adding its meta file

This error occurs when you add an asset directory to your Unity project without adding its companion `.meta` file to git.

Git doesn't keep track of directories explicitly, only files. However, when you add a file in a new directory to git, that directory will be implicitly added.

For example, if you add a new asset file `Assets/Textures/Explosion.png` to git, this implicitly adds the directory `Assets/Textures` if that directory didn't already exist. When this happens you should also add the companion meta file for the new directory `Assets/Textures.meta` .

How to fix this error

Usually when you see this error, the meta file for the directory already exists and you just need to stage it for addition to git with `git add <META-FILENAME>` or your git client's equivalent.

For example `git add Assets/Textures.meta`

How to avoid in the future

When adding assets in new directories to git, make sure you add the meta files for those new directories.

Meta file added without adding its Asset

This error occurs when you add the meta file for an asset in your Unity project to git without adding the asset itself.

For example, if you added a meta file `Assets/Textures/Explosion.png.meta` to git, you should also add its corresponding asset file `Assets/Textures/Explosion.png`

How to fix this error

Usually when you see this error, the asset file already exists and you just need to stage it for addition to git with `git add <ASSET-FILENAME>` or your git client's equivalent command.

For example `git add Assets/Textures/Explosion.png`

How to avoid in the future

When staging new assets for commit to git, make sure you add both the asset **and** its meta file.

Asset deleted without deleting its meta file

This error occurs when you delete an asset in your Unity project from git without deleting its companion `.meta` file.

For example, if you deleted an asset file `Assets/Textures/Explosion.png` from git, you should also delete its companion meta file `Assets/Textures/Explosion.png.meta` so that you don't leave an orphaned meta file in your git repository.

How to fix this error

To fix this error you need to stage the meta file for deletion from git with `git rm <META-FILENAME>` or your git client's equivalent command.

For example `git rm Assets/Textures/Explosion.png.meta`

How to avoid in the future

- Always delete assets within the Unity Editor. This ensures that both the asset and its meta file are deleted together.
- When staging deleted files for commit to git, make sure you include both the asset **and** its meta file.

Directory deleted without deleting its meta file

This error occurs when an asset directory in your Unity project is deleted from git without deleting its companion `.meta` file.

Git doesn't track directories explicitly, only files. However, it does track directories implicitly, so when all of a directory's contents are deleted, that directory is also removed from your git repo.

For example, if you deleted all of the files and sub-directories of the directory `Assets/Textures` from git, you should also delete its companion meta file `Assets/Textures.meta` so that you don't leave an orphaned meta file in your git repository.

How to fix this error

To fix this error you need to stage the directory's meta file for deletion from git with `git rm <META-FILENAME>` or your git client's equivalent command.

For example `git rm Assets/Textures.meta`

How to avoid in the future

- When removing a directory from git, either by deleting its contents or by moving them elsewhere, make sure you delete the directory's companion meta file.
- Avoid creating empty asset directories in your Unity project as they won't be tracked in git.

Meta file deleted without deleting its asset

This error occurs when you delete a meta file in your Unity project from git without deleting the corresponding asset file.

For example, if you deleted a meta file `Assets/Textures/Explosion.png.meta` to git, you should also delete its companion

asset file `Assets/Textures/Explosion.png`

How to fix this error

To fix this error you need to stage the asset for deletion from git with `git rm <ASSET-FILENAME>` or your git client's equivalent command.

For example `git rm Assets/Textures/Explosion.png`

How to avoid in the future

- Always delete assets within the Unity Editor. This ensures that both the asset and its meta file are deleted together.
- When staging deleted files for commit to git, make sure you include both the asset **and** its meta file.

Asset renamed without renaming its meta file

This error occurs when you rename an asset in your Unity project without renaming the corresponding meta file.

For example, if you renamed an asset `Assets/Textures/Explosion.png` to `Assets/Textures/Bang.png` in git, you should also rename its companion meta file `Assets/Textures/Bang.png.meta`

How to fix this error

This error usually occurs because you have forgotten to stage the renamed meta file for commit to git. If this is the case you need to stage the renamed meta file for commit to git with `git add <META-FILENAME>` or your git client's equivalent command.

For example `git rm Assets/Textures/Bang.png.meta`

How to avoid in the future

- Always rename assets within the Unity Editor. This ensures that both the asset and its meta file are renamed together.
- When staging changes for commit to git, make sure you include both the renamed asset **and** its meta file.

Meta file renamed without renaming its meta file

This error occurs when you rename a meta file in your Unity project without renaming the corresponding asset file.

For example, if you renamed a meta file `Assets/Textures/Explosion.png.meta` to `Assets/Textures/Bang.png.meta` in git, you should also rename its companion asset `Assets/Textures/Bang.png`

How to fix this error

To fix this error you need to stage the renamed asset for commit to git with `git add <ASSET-FILENAME>` or your git client's equivalent command.

For example `git rm Assets/Textures/Bang.png`

How to avoid in the future

- Always rename assets within the Unity Editor. This ensures that both the renamed asset and its meta file are renamed together.
- When staging changes for commit to git, make sure you include both the renamed asset **and** its meta file.

Corrupt meta file contents

This error occurs when the contents of meta file become corrupted so that they can no longer be read by Unity.

This error is often caused when git detected conflicts in a meta file during a merge, but those conflicts weren't resolved properly.

When git detects conflicts in a file it writes both versions of the conflicted it to the line like this:

```
fileFormatVersion: 2
<<<<<<< HEAD:Explosion.png.meta
guid: ba4d185bfb5b94901b6da824bfbfe6fa
=======
guid: abc45129ef91683ac929b93458652185
>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:Explosion.png.meta
NativeFormatImporter:
  externalObjects: {}
...
```

In the above example, git detected a conflict on the line starting `guid:` so it wrote both versions of this line to the file, surrounded by markers `<<<<<<<` , `=======` and `>>>>>>>` to draw your attention to the conflict.

Git expects you to resolve the conflict by editing the file to include **either** one version of conflicted lines or the other, **but not both versions**.

Once you have fixed the conflict you then notify git that it has been dealt with by re-adding it with `git add <CONFLICTED-FILENAME>` or your git client's equivalent command.

But, if you re-add the file without resolving the conflicts, the markers remain, making the file unreadable by Unity and triggering this error.

How to fix this error

To fix this error you need to edit the affected meta file in a text editor, removing the conflict markers along with **one version** of the conflicted lines to resolve the conflict.

Continuing with the above example, we might choose to take the remote (HEAD) version of the conflicted lines as follows:

```
fileFormatVersion: 2
guid: ba4d185bfb5b94901b6da824bfbfe6fa
NativeFormatImporter:
  externalObjects: {}
...
```

Now that we have resolved the conflict and removed the conflict markers, the file becomes readable again. Now we can re-add the resolved file to fix the error.

N.B. Whenever you resolve conflicts like this, you should test your changes in the Unity Editor to make sure that everything still works as expected.

How to avoid in the future

When git notifies you of a conflicted meta file during a merge, make sure you fix the conflict **before** re-adding the file it to git.

If you re-add the file without fixing the conflict, the meta file will become broken and unreadable by Unity.

Modified GUID in meta file

Unity keeps track of the assets in your project using Globally Unique Identifiers (GUIDs) which are stored in each asset's meta

file. GUIDs are long hex strings that look something like this `ba4d185bfb5b94901b6da824bfbfe6fa`

For instance, when you reference one asset from another, Unity saves the reference by storing the GUID of the referenced asset.

An advantage of this approach is that you can rename and move the assets within your project without breaking references between them.

For instance, if you rename the asset `Assets/Textures/Explosion.png` to `Assets/Textures/Bang.png` in the Unity Editor, any references to that asset will remain intact.

This error is usually caused when you update an asset by deleting it and then creating a new asset with the same name. When you do this, Unity gives the new asset a different GUID to the old one, because, in Unity's eyes, it is a **completely new asset**, even though it has the same name as the original.

When this happens, the new asset has a different GUID to the one it replaces, so any references to the old asset will become broken.

This is almost never what you want.

How to fix this error

Caution: This fix will lose any changes you have made to the affected assets's settings that you see in the Inspector window.

The easiest way to fix this error is to revert your changes to the asset's meta file with `git checkout <META-FILENAME>` or your git client's equivalent command.

This will restore the GUID to its original value, but it will also **revert any changes you have made to the asset's settings** that you see in the Inspector window.

So, if you apply this fix, you will have to reapply any changes that you have to asset's settings once the meta file has been reverted.

This is a pain and is something we hope to fix with MetaBuddy's automatic fixes feature.

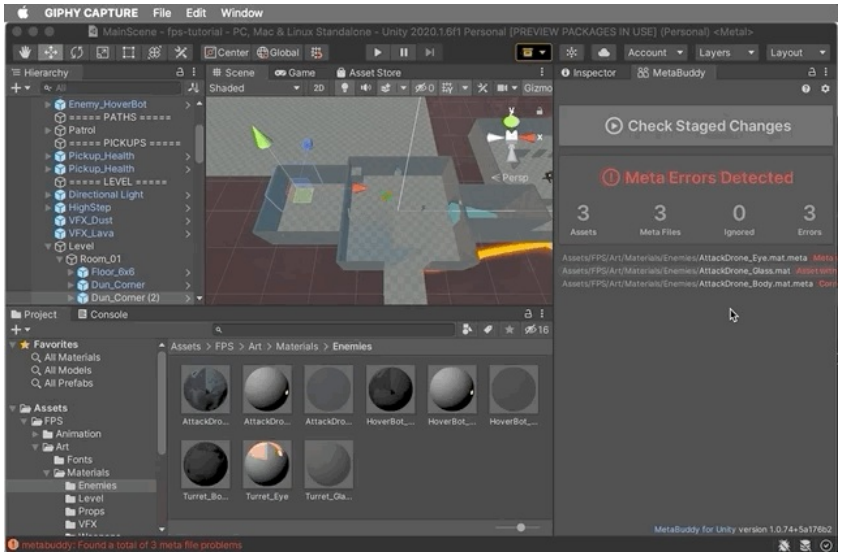
How to avoid in the future

Avoid updating assets by deleting and re-creating them.

Instead use the **Reimport** option on the asset's context menu to update the asset contents without changing its GUID or settings from the Inspector window.

Configuration

MetaBuddy's configuration options are in the **Unity Project Settings** window at **Edit->Project Settings->MetaBuddy**



Configuration file

The MetaBuddy configuration for your project is saved to a human readable YAML file in your project's root directory.

The default name for this configuration file `.metabuddy.yml` When you change the MetaBuddy settings in **Unity Project Settings**, this file will be created if it doesn't already exist.

You can instruct MetaBuddy to look for configuration under a different filename through the `-mb-config` command line option.

We recommend that you commit the configuration file to your project's git repository. That way everyone your team will be working with a consistent configuration.

Hide banner in Console and log files

By default MetaBuddy prints a banner message with version and copyright information to the Unity Console. It looks something like this:

```
metabuddy: MetaBuddy for Unity version 1.0.29.0
Copyright(C) Rate Limited. All rights reserved. https://ratelimited.netlify.app/metabuddy/
```

Enable this option to suppress printing of the banner message.

Ignore files from .gitignore

Git allows you specify which files in your project should be excluded from version control through a file named .gitignore.

When this option is enabled, MetaBuddy will follow git's lead, skipping the checking of files which match the patterns in your project's .gitignore file (if it has one).

This option is enabled by default as it almost always the behavior that want.

Verbose logging

When this option is enabled, MetaBuddy logs more verbosely to the Unity Console and log file.

This option can specified on the command line with `-mb-verbose`.

Run analysis when Unity Editor starts

Tells MetaBuddy to run a check whenever the Unity Editor starts up.

This option can specified on the command line with `-mb-analyse-on-editor-startup`.

Log ignored files to the console

Tells MetaBuddy to log the names of files that were ignored during analysis to the Unity Console and log file.

This option can specified on the command line with `-mb-log-ignored-files`.

Command Line Usage

You can instruct Unity to run MetaBuddy from the command line builds.

This can be useful if you want to run MetaBuddy as part of an unattended build process, for instance as part of a Continuous Integration (CI) job on your build server.

Starting MetaBuddy from the command line

To run a MetaBuddy check from the command line, use Unity's `-executeMethod` command line option like this:

```
Unity -projectPath <YourProject> -executeMethod MetaBuddy.CLI.Run
```

When MetaBuddy is started from the command line in this way, it exits Unity as soon as it has finished checking your project. It signals the success or failure of the checking process through the return code returned by Unity.

You can arrange for your CI system to fail any build containing meta file errors by inspecting this return code.

A code of 0 will be returned if no meta file errors were found. If one or more errors was found during checking, a non-zero error code will be returned.

Command line options

A selection of the configuration options available through MetaBuddy's **Unity Project Settings** page can also be specified on the command line like this:

```
Unity -projectPath <YourProject> -executeMethod MetaBuddy.CLI.Run -mb-config config-for-ci.yml -mb-verbose
```

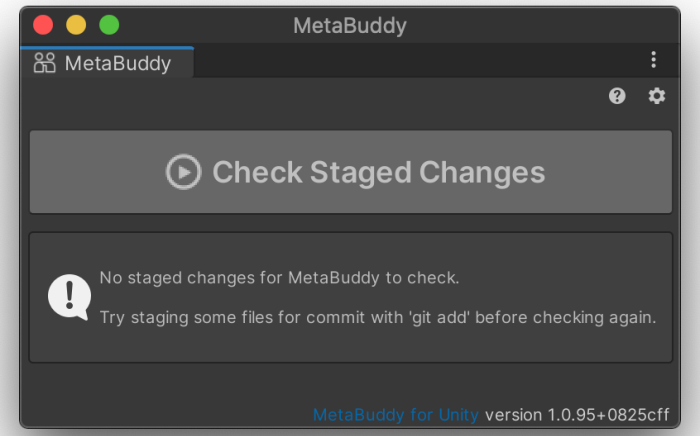
In the above example, we tell MetaBuddy tell to load its configuration from a file named `config-for-ci.yml` instead of the default `.metabuddy.yml` We also specify that MetaBuddy should log verbosely to the Unity log file.

Options specified on the command line override those loaded from the configuration file. You can use the verbose logging option `-mb-verbose` to have MetaBuddy print all of its configuration options to the log before checking commences.

FAQ and Troubleshooting

No staged changes for MetaBuddy to check

When you see this message everything is working fine, but there's nothing for MetaBuddy to do yet.

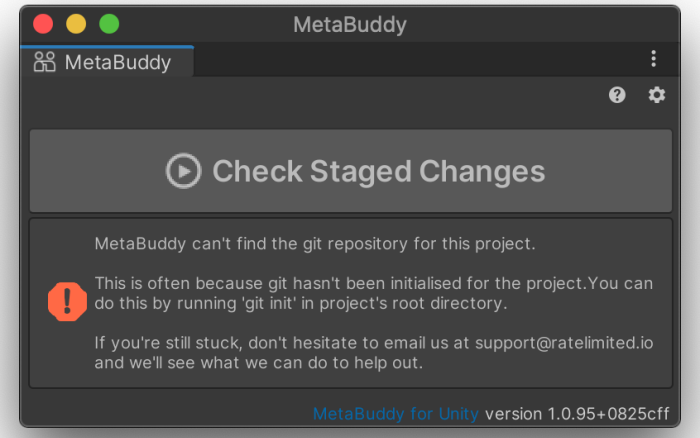


MetaBuddy checks the changes that will be included in your next commit to git. Git calls these **staged** changes.

By checking only staged changes, MetaBuddy ensures that each individual commit that you make to git is complete and correct, without bothering you with false warnings for files that you aren't intending to commit.

As soon as you stage some files with `git add` or your favorite git client, MetaBuddy will pick them up and this message will disappear.

MetaBuddy can't find the git repository for this project



When you see this, it almost always means that your project isn't in a git repository yet.

The most basic way to setup a repository is to run `git init` from your project's root directory. But, there's a little bit more to do in order to get git working smoothly with Unity.

You can find a step-by-step guide to setting up a git repository for your project here.

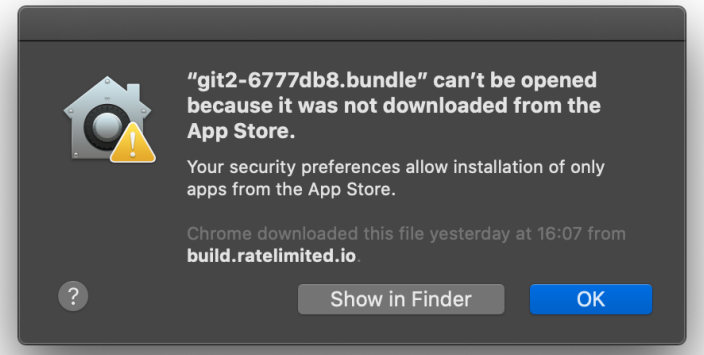
How does MetaBuddy find my project's git repository?

MetaBuddy finds your project's repository by searching for a folder named `.git` in the root of your project. The .git folder is where git stores all of its version control information.

If MetaBuddy doesn't find a `.git` directory in the root of your project, it searches each of the directories above your project in turn.

This is useful for cases where your Unity project isn't in the root of your repository. For instance, when you have several Unity projects alongside each other within the same git repository.

macOS tells me that "git2-6777db8.bundle can't be opened because it was not downloaded from the App Store"?



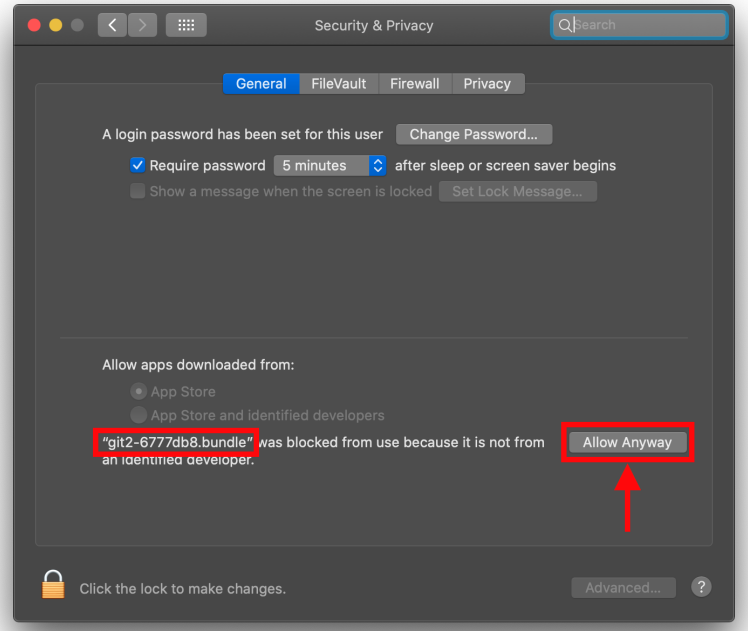
This is caused by the GateKeeper feature in macOS which checks that all applications have been **downloaded from the Apple App Store** or have been **signed by a Certified Apple Developer and notarized by Apple**.

It doesn't make much sense to distribute a Unity Asset like MetaBuddy through the Apple App Store, so MetaBuddy uses the second option of having its code signed and notarized by Apple.

If you see this message it usually means that you have GateKeeper set to super-strict mode where only applications from the Apple App Store are allowed. There are two ways you can fix this.

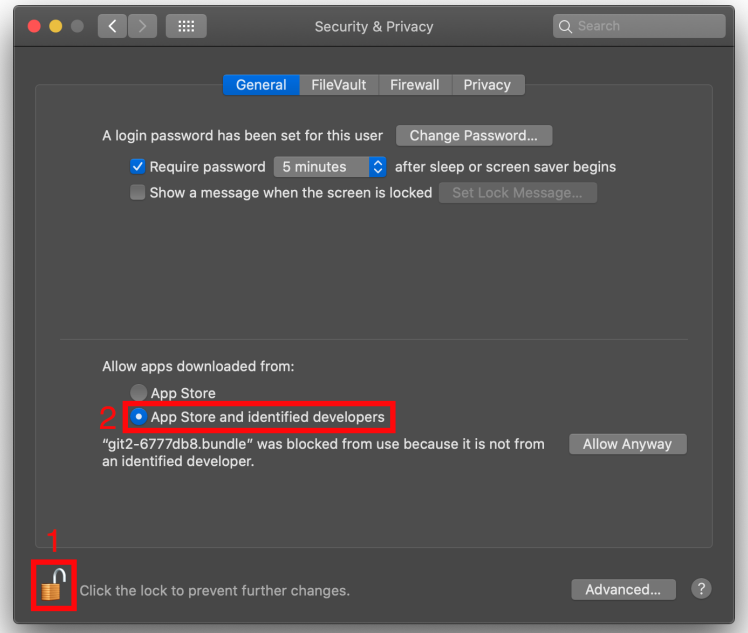
Option 1. Create a one-off exception for MetaBuddy

- Choose **System Preferences** from the **Apple Menu**.
- Click the **Security & Privacy** icon.
- Choose the **General** tab.
- You will see a message telling you that git2-6777db8.bundle came from an unidentified developer.
- Click the **Allow Anyway** button to the right of the message.
- Restart Unity and reopen your project.



Option 2. Allow applications from identified developers (even if they didn't come from the Apple App Store)

- Choose **System Preferences** from the **Apple Menu**.
- Click the **Security & Privacy** icon.
- Choose the **General** tab.
- Click the **Padlock** icon in the bottom left of the window.
- Enter your password when prompted.
- Select the **App Store and identified developers** option.
- Restart Unity and reopen your project.



Support

Got questions? Need help?

- Drop us an email at support@ratelimited.io
- Raise an issue in our GitHub repository.
- Fill in our Contact Form
- Chat to our team on the MetaBuddy Discord Server