Using Git with Unity

Git is a popular free and open source distributed version control system, created in 2005 by Linus Torvald.

These instructions will guide you through process of setting up git and Unity in a way that will help you keep your sanity. It's not a perfect solution, but it works fairly well. If you are looking for an introduction to git, check out these tutorials by Atlassian instead.

What will I get from this?

If you follow all the instructions in this document, you will be able to merge and diff Unity scenes, assets and prefabs with git. Consider the following scenario: - On your `master` branch, you have a Prefab `MyPrefab` with a position of (0,0,0). - Alice, a teammate, has commited `MyPrefab` with a position of (4,4,0) in branch `b440`. - Bob, another teammate, has commited `MyPrefab` with a position iof (0,5,5) in branch `b055`.

Our repository looks something like this:

```
* MyPrefab (4,4,0) - Alice (b440)
| * MyPrefab (0,5,5) - Bob (b055)
|/
* MyPrefab (0,0,0) - Me (HEAD -> master) <-- You are here
```

We want to merge Alice's work.

```
$ git merge b440
Updating 0505350..28676ca
Fast-forward
 Assets/MyPrefab.prefab | 12 ++++++++++--
 1 file changed, 10 insertions(+), 2 deletions(-)
```

Git performed what it calls a "Fast-forward" merge. It reliazed the master branch is fully merged into the branch b440 and updated the code without the need to do any actual merging.

Now our repository looks somethig like this:

```
* MyPrefab (4,4,0) - Alice (HEAD -> master, b440) <-- You are here
| * MyPrefab (0,5,5) - Bob (b055)
|/
```

```
* MyPrefab (0,0,0) - Me
```

Time to merge Bob's work.

```
$ git merge b055
warning: Cannot merge binary files: Assets/MyPrefab.prefab (HEAD vs. b055)
Auto-merging Assets/MyPrefab.prefab
CONFLICT (content): Merge conflict in Assets/MyPrefab.prefab
Automatic merge failed; fix conflicts and then commit the result.
```

Because both Alice and Bob have modified the position of the transform componant of the prefab, git doesn't know which position to keep.

We need to use *UnityYAMLMerge* (which we'll talk about later) to manually resolve this conflict. Let's do it!

```
$ git mergetool
Merging:
Assets/MyPrefab.prefab

Normal merge conflict for 'Assets/MyPrefab.prefab':
  {local}: modified file
  {remote}: modified file
Conflicts:
Left  418756.Transform.m_LocalPosition.y change to 5
Right 418756.Transform.m_LocalPosition.y change to 4
```

The last two lines are what's really important. It tells us that there is a conflict on the `y` componant of `MyPrefab`, and that the two conflicting values are `4` and `5`. Notice how the `x` and `z` components were merged automatically because there was no conflict (Alice set x,y,z to 4,4,0 and Bob set x,y,z to 0,5,5).

Our favorite merge tool will now open (I chose to use KDiff3):

Select the changes you want to keep (I will choose 4.5 to make everyone happy), save, and close the window. Git will tell you that you need to commit to conclude the merge.

```
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
```

```
        modified:   Assets/MyPrefab.prefab
```

Let's finalze our merge and allow git to create a merge commit.

```
$ git commit
[master 98d7881] Merge branch 'b055'
```

Behold! Our three branches are now all merged together.

```
*   MyPrefab (4,4.5,5) - Me (HEAD -> master) <-- You are here
|\
| * MyPrefab (4,4,0) - Alice (b440)
* | MyPrefab (0,5,5) - Bob (b055)
|/
* MyPrefab (0,0,0) - Me
```

The rest of this page will explain to you how to configure git and Unity to achieve the same result.

Configure Unity

Smart Merge

http://docs.unity3d.com/Manual/SmartMerge.html

> Unity incorporates a tool called UnityYAMLMerge that can merge scene and prefab files in a semantically correct way. The tool can be accessed from the command line and is also available to third party version control software.

Unity 5 comes with a tool called *UnityYAMLMerge* that can automatically merge scenes, assets and prefabs stored in YAML format. If there is a conflict, it will open a standard merge tool (such as KDiff3 or Meld) for a 3-way merge where *BASE*, *LOCAL* and *REMOTE* will all have been preprocessed.

Our objective is to configure git and Unity to take advantage of this tool.

Custom Merge Tool (optional)

You can configure *UnityYAMLMerge* to use a merge tool of your choice. Edit the file `mergespecfile.txt` accordingly. For instance, to enable KDiff3, change the following two lines...

```
unity use "%programs%/KDiff3/kdiff3.exe" "%b" "%l" "%r" -o "%d"
prefab use "%programs%/KDiff3/kdiff3.exe" "%b" "%l" "%r" -o "%d"
```

...and add the following line at the begining of the section 'Default fallbacks for unknown files'.

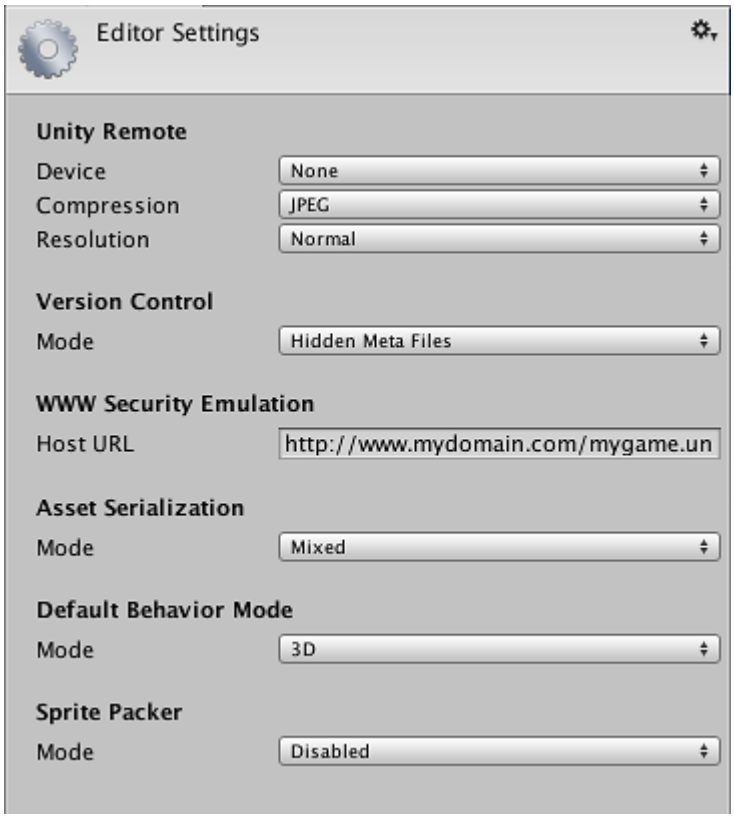
```
* use "%programs%/KDiff3/kdiff3.exe" "%b" "%l" "%r" -o "%d"
```

Asset Serialization

http://docs.unity3d.com/Manual/class-EditorManager.html

> To assist with version control merges, Unity can store scene files in a text-based format (see the text scene format pages for further details). If scene merges will not be performed then Unity can store scenes in a more space efficient binary format or allow both text and binary scene files to exist at the same time.

In order for the *UnityYAMLMerge* tool to work, assets must be stored in YAML format. Go into Edit > Project Settings > Editor, set the `Asset Serialization` option to `Force Text`.



Configure Git

Git mergetool

Now that everything is setup on the Unity side, we need to tell git to use the proper merge tool.

Add the following to your .git/config:

```
[merge]
    tool = unityyamlmerge
```

```
[mergetool "unityyamlmerge"]
    trustExitCode = false
    keepTemporaries = true
    keepBackup = false
    path = 'C:\\Program Files\\Unity\\Editor\\Data\\Tools\\UnityYAMLMerge.exe'
    cmd = 'C:\\Program Files\\Unity\\Editor\\Data\\Tools\\UnityYAMLMerge.exe' merge
-p "$BASE" "$REMOTE" "$LOCAL" "$MERGED"
```

Edit the paths to match those on your environment.

Next, we want to prevent git from trying to merge some types of file on its own by declaring them as binary files.

To do this, put the following in your .gitattributes:

```
*.unity binary
*.prefab binary
*.asset binary
```

The good news is that *UnityYAMLMerge* will do all the merging for us. The bad news if that the file will be treated as on opaque blob as if it was still in binary format.

Human-redable diffs (optional)

We can use a simple textconv script to retain basic, human-readable diffs.

Add this to your .git/config:

```
[diff "unity"]
      textconv='C:\\Users\\petit_v\\unityYamlTextConv.py'
```

Edit the paths to match those on your environment.

Then, edit your .gitattributes to read (note the added `diff=unity`):

```
*.unity binary diff=unity
*.prefab binary diff=unity
*.asset binary diff=unity
```

An example of a textconv script, `unityYamlTextConv.py`, could be:

```
#!/usr/bin/env python

import sys
import yaml
import pprint
```

```
if len(sys.argv) < 2:
    sys.exit(-1)

def tag_unity3d_com_ctor(loader, tag_suffix, node):
    values = loader.construct_mapping(node, deep=True)
    if 'Prefab' in values:
        if 'm_Modification' in values['Prefab']:
            del values['Prefab']['m_Modification']
    return values

class UnityParser(yaml.parser.Parser):
    DEFAULT_TAGS = {u'!u!': u'tag:unity3d.com,2011'}
    DEFAULT_TAGS.update(yaml.parser.Parser.DEFAULT_TAGS)

class UnityLoader(yaml.reader.Reader, yaml.scanner.Scanner, UnityParser,
yaml.composer.Composer, yaml.constructor.Constructor, yaml.resolver.Resolver):
    def __init__(self, stream):
        yaml.reader.Reader.__init__(self, stream)
        yaml.scanner.Scanner.__init__(self)
        UnityParser.__init__(self)
        yaml.composer.Composer.__init__(self)
        yaml.constructor.Constructor.__init__(self)
        yaml.resolver.Resolver.__init__(self)

UnityLoader.add_multi_constructor('tag:unity3d.com', tag_unity3d_com_ctor)
with open(sys.argv[1], 'r') as stream:
    docs = yaml.load_all(stream, Loader=UnityLoader)
    for doc in docs:
        pprint.pprint(doc, width=120, indent=1, depth=6)
```

Now it's possible to see what changed inside of a prefab, asset or scene:

```
33  BoxCollider2D:
34  ··m_ObjectHideFlags:·1
35  ··m_PrefabParentObject:·{fileID:·0}
36  ··m_PrefabInternal:·{fileID:·100100000}
37  ··m_GameObject:·{fileID:·132702}
38  ··m_Enabled:·1
39  ··m_Density:·1
40  ··m_Material:·{fileID:·0}
41  ··m_IsTrigger:·1
42  ··m_UsedByEffector:·0
43  ··m_Offset:·{x:·0,·y:·0}
44  ··serializedVersion:·2
45  ··m_Size:·{x:·1,·y:·1}
```

```
33  BoxCollider2D:
34  ··m_ObjectHideFlags:·1
35  ··m_PrefabParentObject:·{fileID:·0}
36  ··m_PrefabInternal:·{fileID:·100100000}
37  ··m_GameObject:·{fileID:·132702}
38  ··m_Enabled:·1
39  ··m_Density:·1
40  ··m_Material:·{fileID:·0}
41  ··m_IsTrigger:·1
42  ··m_UsedByEffector:·0
43  ··m_Offset:·{x:·0,·y:·0}
44  ··serializedVersion:·2
45  ··m_Size:·{x:·25,·y:·25}
```

[rant]

You may have noticed that the script above will not display the `m_Modification` field inside of a Prefab. Apparently, someone at Unity thought it was a good idea to store a *complete* list of all the modifications to a prefab inside of said prefab. **Why?**... I use a version constrol system whose sole purpose is to keep track of file changes, I don't need or want any of this.

If you make a change to a Prefab in the Editor and then Ctrl-Z, you can get yourself into a situation like this one:

```
$ git status
```

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Assets/prefabs/Asteriod.prefab

no changes added to commit (use "git add" and/or "git commit -a")

$ git diff

$
```

This is because `m_Modification` has been altered, even though the prefab is still the same in every way. At this point, you should `git chekout` the file back, unless you want to introduce needless, senseless non fast-forward merges in your history. In fact, the *UnityYAMLMerge* tool seems to be ignoring this section entirely.

This is **complete nonsense**, this is **madness**, and it makes me **sad**.
[/rant]

.gitignore (optional)

You may use the following .gitinore file. Up-to-date definitions can be found on github.

```
#
# == Windows ==
#

# Windows image file caches
Thumbs.db
ehthumbs.db

# Folder config file
Desktop.ini

# Recycle Bin used on file shares
$RECYCLE.BIN/

# Windows Installer files
*.cab
*.msi
*.msm
*.msp

# Windows shortcuts
*.lnk

#
# == Unity ==
#

/[Ll]ibrary/
```

```
/[Tt]emp/
/[Oo]bj/
/[Bb]uild/
/[Bb]uilds/
/Assets/AssetStoreTools*

# Autogenerated VS/MD solution and project files
ExportedObj/
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
*.user
*.userprefs
*.pidb
*.booproj
*.svd


# Unity3D generated meta files
*.pidb.meta

# Unity3D Generated File On Crash Reports
sysinfo.txt

# Builds
*.apk
*.unitypackage
```

Service annoucement: remember to commit all your **.meta** (another great feature), or kitten *will* die.

Vianney,
:wq