1. 什么是SVN Hook

Hook，顾名思义，译为钩子。所谓的SVN Hook，即用户在对SVN仓库进行操作的时候，当特定的事务发生时，对应的Hook就会被调用，Hook就是特定事件的处理函数。

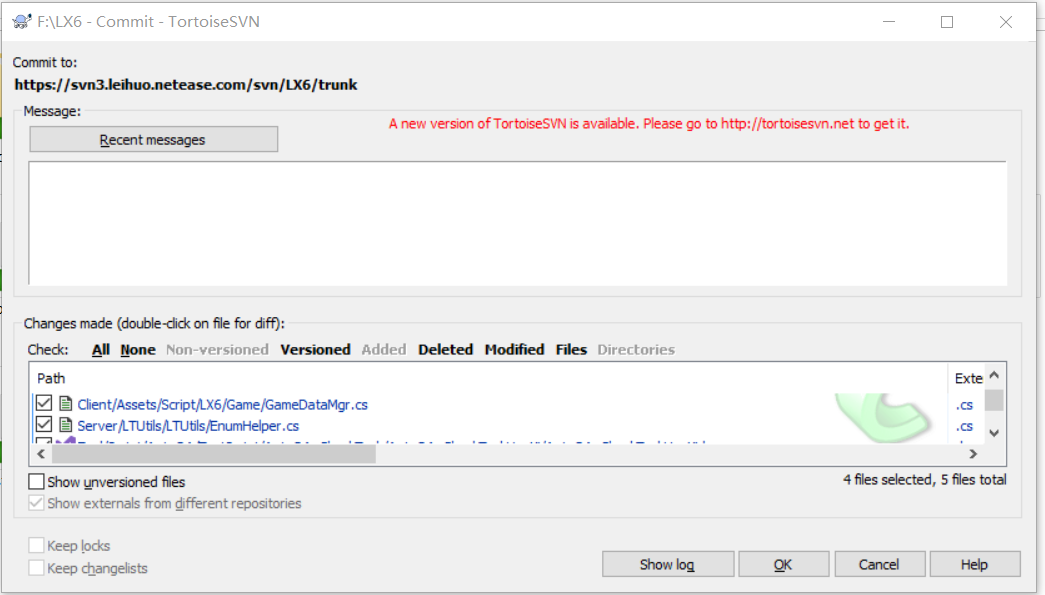
1. 常用的SVN客户端Hook种类
   1. 客户端提交Hook
      1. Start-commit

在按下SVN Commit之后，提交对话框显示之前被调用。 可以根据用户名对提交权限进行控制，多用于对svn版本功能check的检查。



* + 1. Pre-commit

在用户单击提交对话框中的确定按钮，并且实际的提交过程开始后被调用。此时提交事务已生成。此时可以获取到详细的提交内容。是我们平时最为常用的一种钩子类型。

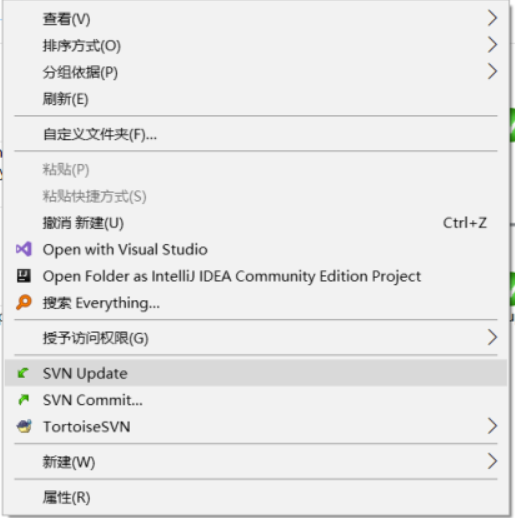


* + 1. Post-commit

提交事务成功完成后会进行调用。多用于对数据进行备份或者是进行一些特定文件提交后的分析与提醒，或者进行一些自动构建等。

* 1. 客户端更新Hook
     1. Start-update

在用户按下SVN Update之后，更新对话框显示之前会被调用。



* + 1. Pre-update

在弹出更新对话框，真正开始进行更新（update or switch）之前进行调用。

* + 1. Post-update

在更新（update, switch or checkout finishes）之后，无论成功与否，都会进行调用。

* 1. 不同种类的Hook对应的参数

PATH：临时文件的路径，其中有提交文件的所有路径。

DEPTH：提交/更新的深度。

MESSAGEFILE：在Start-commit执行完毕后生成，包含了日志信息。

REVISION：更新或提交完成后的版本库的版本。

ERROR：包含错误信息的文件的路径， 如果没有错误的话，文件将是空的。

CWD：脚本正在运行的工作目录,一般为当前SVN仓库的根目录。

RESULTPATH：临时文件的路径，类似于PATH

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | PATH | DEPTH | MESSAGEFILE | REVISION | ERROR | CWD | RESULTPATH |
| Start-commit | **√** |  | **√** |  |  | **√** |  |
| Pre-commit | **√** | **√** | **√** |  |  | **√** |  |
| Post-commit | **√** | **√** | **√** | **√** | **√** | **√** |  |
| Start-update | **√** |  |  |  |  | **√** |  |
| Pre-update | **√** | **√** |  | **√** |  | **√** |  |
| Post-update | **√** | **√** |  | **√** | **√** | **√** | **√** |

1. 组内客户端Pre-commit详情介绍

3.1 客户端Pre-commit的意义

在项目过程中，最常用的便是客户端Pre-commit。

当项目组人员进行提交操作时，经常会因为手误、规则不清等原因导致一些本来约定好的规则被破坏，等提交后由QA进行测试时才会发现这些错误，然后由QA进行反馈并开单，由提交者再去进行修复。这样无疑会对人力造成极大的浪费，不利于项目高效地开发。更有甚者会提交到一些非常严重的阻碍性的bug，导致项目开发阶段停滞。因此，在提交时进行一系列的检查，使得原本该出现的问题防患于未然，才是最高效的解决这一问题的方式。

3.2 提交格式的检查

提交时获取MESSAGEFILE，里面含有提交时的信息

3.2.1 提交需关联易协作

本人项目组是使用易协作进行任务单的跟进和测试的，在测试BUG、需求等任务单时，为了方便查看该任务单相关的提交内容，需要要求组内人员在提交时注明提交内容对应的任务单号。这样在测试人员进行测试时，可以一目了然的查看提交者具体修改了哪些内容，快速查看提交者有无提交不规范、错误提交的情况。

3.2.2 代码类提交需要增加review人员名称

本人项目组使用C#结合lua进行游戏内容的开发，俗话说三个臭皮匠赛过诸葛亮，开发者在进行代码编写时难免会出现考虑不周的情况，因此在提交前找一个同事或者上级主管对其提交内容进行review可以大大提高代码的健全度。同时由于在提交内容中增加了review name，当提交有问题时可以方便定位提交者和进行review的人。

3.3 提交文件路径、文件名、文件大小的检查

提交时获取PATH，对其进行格式的检查

3.3.1 提交文件路径长度的限制

本人项目组内遇到过一次很奇葩的情况，在构造bunlde资源时，有一个文件的path太长了，导致bundle构建报错了。Windows在.netframe work框架里的删除目录的方法中，有些地方写死了最大长度为256个字符，超过了则会导致路径被截断，从而导致bundle资源无法正常构造。因此增加了对提交路径长度的限制，如果长度超过限制长度，则直接拒绝其进行提交

3.3.2 提交文件名的限制

由于制作规范问题，在策划、美术等职能进行资源的提交时，常常会出现一些命名不规范的情况，比如进行提交文件中包含了中文命名、特殊符号命名、包含空格的命名等。根据常用的变量命名规则，在提交检查中增加了队文件名的检查，名称必须为数字、字母、下划线中的一种或多种的组合，且不能为数字开头。

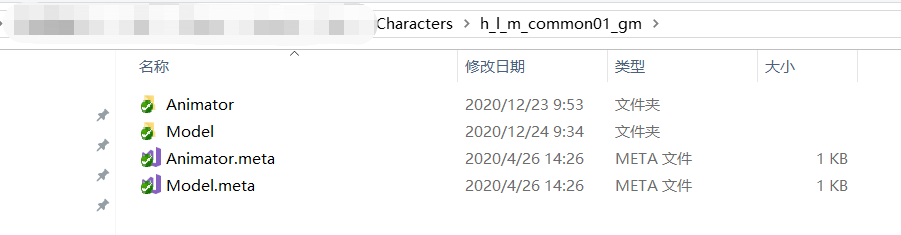
3.3.3 提交文件大小的限制

在资源制作中，由于某些错误配置会导致一个资源出现几十、上百M的情况，这些文件在构造bundle时会非常拖慢构造速度，并且导致真机包体大小失控。为了避免这种情况的发生，在提交检查中增加了单个文件大小不能超过50M的限制。同时为了避免无效资源的提交，限制了字节数为0 的空文件提交。

3.4 美术资源规范性的检查

3.4.1 资源种类的限制

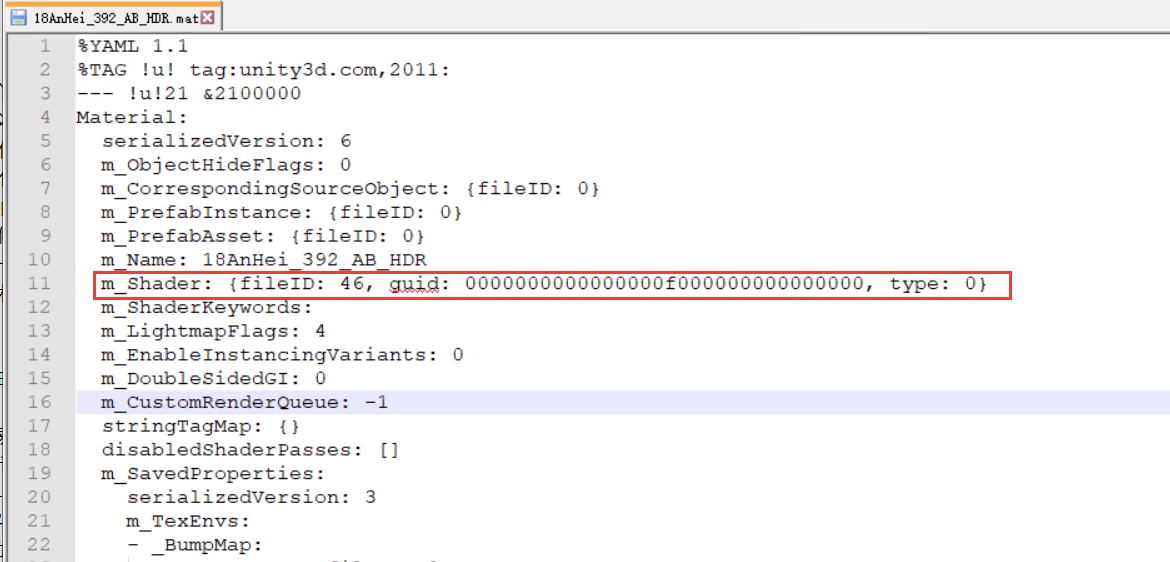
本人所在的项目工程因为文件夹层级众多，难免会出现误提交一些本不属于该文件夹的文件。拿角色资源路径举例，在Charaters文件夹下包含了N个角色对应的资源文件夹，每个文件夹里面又包含了Animator和Model两个文件夹。Animator中只要求提交Animator角色动作相关的资源，而Model中则只要求提交角色相关的贴图、材质以及模型资源。其他资源则一律拒绝提交。



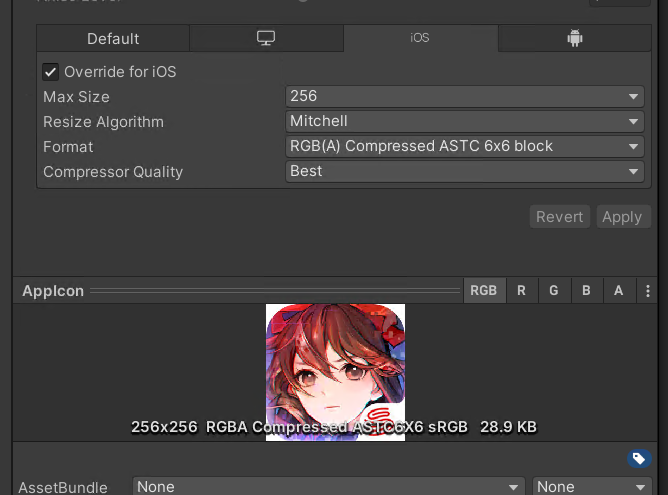
3.4.2 资源内容的限制

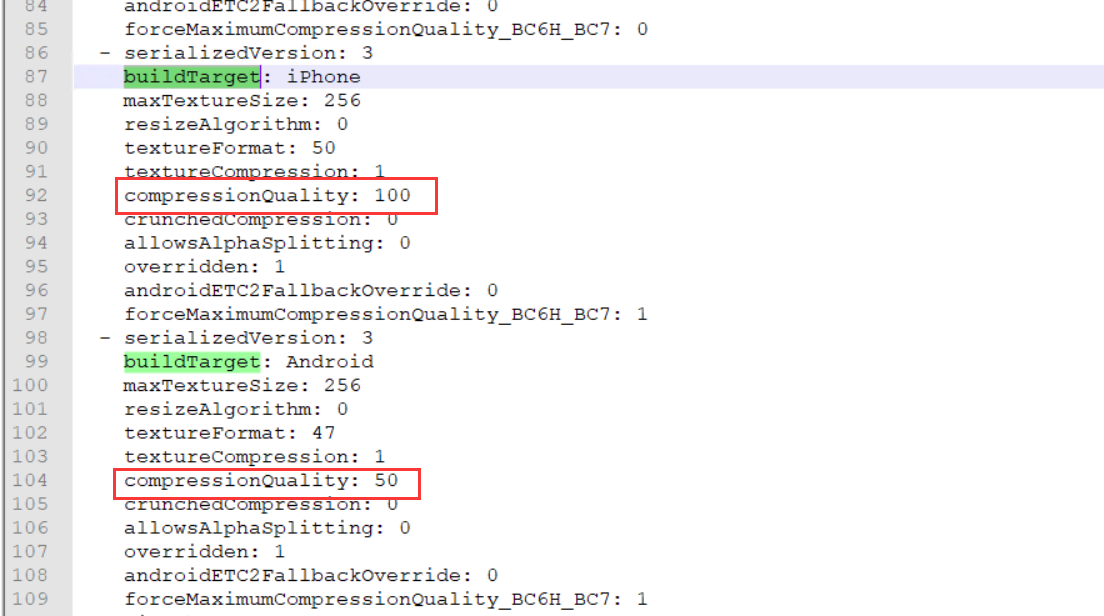
美术和策划等职能通常会通过制定好的规范来进行美术资源的制作，拿材质球来举例，Unity引擎会将默认的材质球的shader设置为Standard材质，而在我们项目组中是必须要求材质球使用项目组内制作好的shader。当出现材质球忘记进行修改从而使用了默认standard材质时，就需要在提交时进行提前量的检查。通过文本文件的格式来打开材质球，发现Standard的shader为

m\_Shader: {fileID: 46, guid: 0000000000000000f000000000000000, type: 0}，因此只需获取PATH中的.mat文件，并读取其中文本包含如上guid时禁止提交。



另一个例子，在进行贴图资源的自作时，需要对其进行多平台的压缩格式的设置。在IOS平台一般为ASTC，Quality为Best。而Android平台一般为和IOS一致的ASTC（现在能成功运行本人所在项目的安卓机基本全都支持ASTC了），Quality为Normal 或者 Best皆可。压缩格式会影响到不同平台上贴图纹理的显示，因此保证压缩格式的正确也成为了平常非常重要的内容。而贴图相关的设置是存储在对应的meta文件中，因此在提交时需要检查其对应的meta文件的内容。分别查看buildTarget: iPhone和buildTarget: Android中的textureFormat以及compressionQuality。

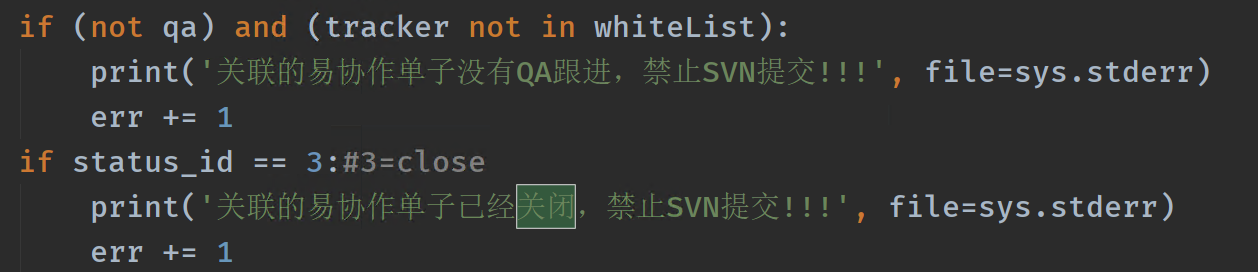




3.5 与易协作的交互

3.5.1 提交关联的任务单必须要有跟进QA

前文有说到，在提交时要求提交信息必须标明关联的易协作任务单。为了避免出现使用一些自主任务、调研等无QA测试的单子提交很多项目资源相关的改动的情况，在提交检查时增加了对易协作单子是否有跟进QA的检查，当发现单子无QA跟进，或者单子是已经验收完成处于关闭状态的单子时，禁止提交。这样可以保证每一个提交一定是有QA来测试的。



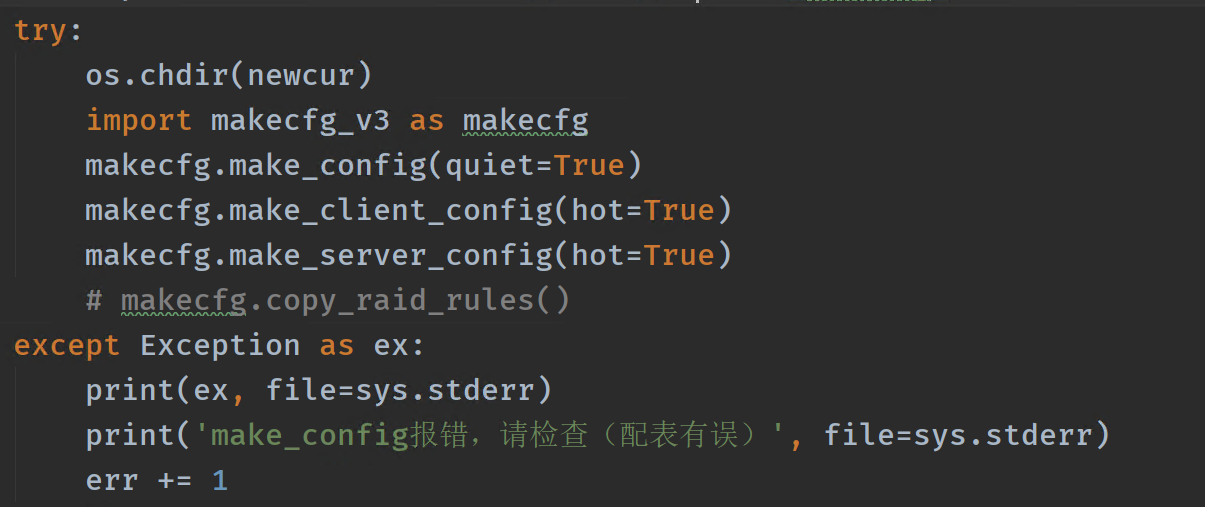
3.5.2 提交人身上Bug数目控制

对于QA来说特别关注的数据里必然会有剩余的未修的BUG数目，而策划、程序则经常会陷于众多需求的海洋中无法自拔，忘记了身上嗷嗷待哺的BUG。因此为了避免出现策划、程序身上堆积十几二十个BUG的情况，本人所在的项目组会对提交时进行提交人身上Bug数目的检查。当提交时会获取提交信息里的单号，如果这个单子是非BUG单，则去拿此人身上的BUG数目，当BUG数目大于8个时，禁止其提交这个单子。但是有些时候还是会有一些紧急需求的情况，当程序有些紧急需求需要立即提交时，这个检查就很尴尬了，需要对这些人加白名单。由于客户端Commit hook的特殊性，如果在Commit Hook代码里进行修改无疑非常繁琐，因此可以通过web的方式去处理这些白名单，做到对提交者无感地更新白名单。并可以将这些数据定期公示到项目组大群中，让项目组成员对此有足够的了解和监督。



3.6 配表规范性检查

在本人所在的项目中，配表的重要性是非常高的。服务端工程编译，以及客户端对配表id的引用，当配表出现服务端编译报错时是会出现严重的项目开发阻碍，因此如果涉及到了配表的提交时，提前进行配表数据编译是非常重要的，既可以避免阻碍项目组其他成员的开发进度，又为策划省去了可乐的惩罚。【不是。



3.7 lua代码检查

本人所在的项目，客户端代码中lua占据很大的比例，lua出现编译报错时会导致打包报错、客户端功能烂掉等问题，因此在本人所在的项目通过使用google推出的一款针对lua静态检查的开源工具-luacheck，来进行lua代码的代码编译检查。

<https://github.com/mpeterv/luacheck>

1. 组内客户端Post-commit详情介绍
   1. 资源修改提醒

在资源提交成功之后，如果有比较关注的资源被改动了，可以通过Post-commit来进行popo通知。

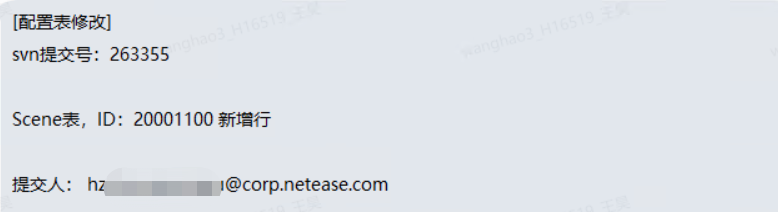


场景内NPC位置修改时，输出对NPC位置改变所影响的任务并通知QA，可以让相关QA即使对改动有了解并尽早进行测试，避免NPC位置改动后造成任务流程卡死。



* 1. 配表修改提醒

场景配表新增时，可以及时了解到相关改动，并可以对新增的场景进行测试。



1. 遇到的问题
   1. 本地客户端Hook脚本未及时更新

由于ClientSide Commit Hook脚本是放在项目文件中，属于本地脚本。当我们对其进行一些修改时，假如其他脚本的使用者一直不对其进行更新的话，会导致我们修改的脚本不会及时起作用。如何保证提交时本地客户端Hook脚本的版本最新以及内容足够完整，还是需要研究的问题。

* 1. 提交检查耗时过长

由于ClientSide PreCommit Hook脚本涉及到很多文件的读写操作，当美术、策划等同学进行大批量的提交或者修改时，几十上百个文件的检查耗时是非常久的。目前需要增加对提交者本地在提交时的每个检查函数运行耗时的统计，生成相关的耗时热点图，对耗时过高的检查及时进行优化。