Confirm a file in source control

This document describes how to confirm (or refute) that the version of a file in an environment matches a file in source control. We propose tagging the commit released to production. We expect the the version of the file in production to be the version in that commit. History makes it easy to see which lines changed. We can confirm a given file matches by verifying the changes are applied.

We discuss techniques using git in VSCode. The process is similar using SVN clients.

We'll use psidev01 in lieu of production for demostration purposes.

Current process

SVN uses keyword substution to replace values like author, last change date, and revision number in the file system version. These values are updated on commit. Let's examine an example file.

`dlgGAGateHouseExitNGLO.spb`

Go to the database (psidev01) and open the package body. The revision keyword let's us know that this file matches the version commited with SVN revision 3380.

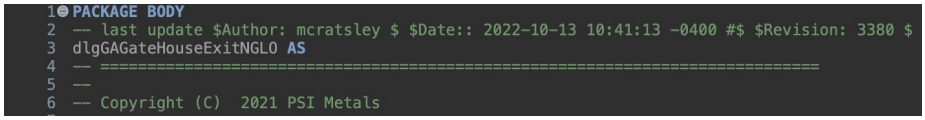

Figure 1: SVN 3380 dlgGAGateHouseExitNGLO.spb header

We can confirm that by going to SVN and checking the log.

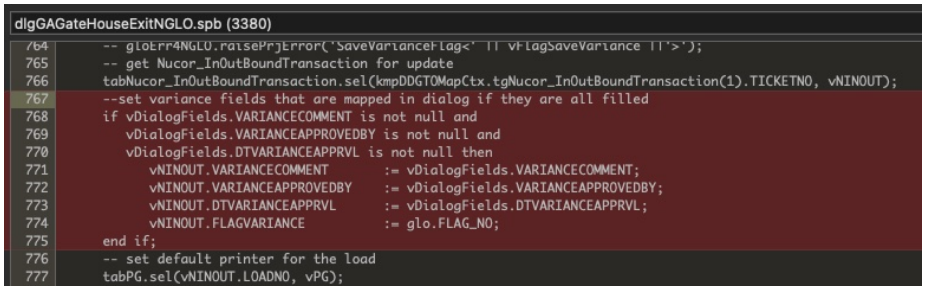

Figure 2: SVN 3380 dlgGAGateHouseExitNGLO.spb

SVN rev 3380 dlgGAGateHouseExitNGLO.spb, first change is from line 767 to 775.

```
764       -- gloErr4NGLO.raisePrjError('SaveVarianceFlag<' || vFlagSaveVariance ||'>');
765       -- get Nucor_InOutBoundTransaction for update
766       tabNucor_InOutBoundTransaction.sel(kmpDDGTOMapCtx.tgNucor_InOutBoundTransaction(1).TICKETNO, vNINOUT);
767       --set variance fields that are mapped in dialog if they are all filled
768       if vDialogFields.VARIANCECOMMENT is not null and
769          vDialogFields.VARIANCEAPPROVEDBY is not null and
770          vDialogFields.DTVARIANCEAPPRVL is not null then
771             vNINOUT.VARIANCECOMMENT       := vDialogFields.VARIANCECOMMENT;
772             vNINOUT.VARIANCEAPPROVEDBY    := vDialogFields.VARIANCEAPPROVEDBY;
773             vNINOUT.DTVARIANCEAPPRVL      := vDialogFields.DTVARIANCEAPPRVL;
774             vNINOUT.FLAGVARIANCE          := glo.FLAG_NO;
775             entSHPLoadHistoryNGLO.addEvent(
776               pi_LoadHistEventID => entSHPLoadHistoryNGLO.LOAD_VARIANCE_APPROVED,
777               pi_PG_ID => vNINOUT.LoadNO,
778               pi_TMBez => vNINOUT.MOTID,
779               pi_VarianceApprovedBy => vNINOUT.VARIANCEAPPROVEDBY
780             );
781       end if;
782       -- set default printer for the load
783       tabPG.sel(vNINOUT.LOADNO, vPG);
```

Figure 3: Database 3380 dlgGAGateHouseExitNGLO body

Lets double check the file on the database to confirm.

The database version marked as rev 3380 ***does not match!*** It is marked as rev 3380 but has a call to `entSHPLoadHistoryNGLO.addEvent()` which does not exist in the SVN rev 3380. The actual version is probably SVN rev 3431.

dlgGAGateHouseExitNGLO.spb (3431)

```
764       -- gloErr4NGLO.raisePrjError('SaveVarianceFlag<' || vFlagSaveVariance ||'>');
765       -- get Nucor_InOutBoundTransaction for update
766       tabNucor_InOutBoundTransaction.sel(kmpDDGTOMapCtx.tgNucor_InOutBoundTransaction(1).TICKETNO, vNINOUT);
767       --set variance fields that are mapped in dialog if they are all filled
768       if vDialogFields.VARIANCECOMMENT is not null and
769          vDialogFields.VARIANCEAPPROVEDBY is not null and
770          vDialogFields.DTVARIANCEAPPRVL is not null then
771             vNINOUT.VARIANCECOMMENT       := vDialogFields.VARIANCECOMMENT;
772             vNINOUT.VARIANCEAPPROVEDBY    := vDialogFields.VARIANCEAPPROVEDBY;
773             vNINOUT.DTVARIANCEAPPRVL      := vDialogFields.DTVARIANCEAPPRVL;
774             vNINOUT.FLAGVARIANCE          := glo.FLAG_NO;
775             entSHPLoadHistoryNGLO.addEvent(
776               pi_LoadHistEventID => entSHPLoadHistoryNGLO.LOAD_VARIANCE_APPROVED,
777               pi_PG_ID => vNINOUT.LoadNO,
778               pi_TMBez => vNINOUT.MOTID,
779               pi_VarianceApprovedBy => vNINOUT.VARIANCEAPPROVEDBY
780             );
781       end if;
782       -- set default printer for the load
783       tabPG.sel(vNINOUT.LOADNO, vPG);
```

Figure 4: SVN 3431 dlgGAGateHouseExitNGLO.spb

SVN rev 3431 dlgGAGateHouseExitNGLO.spb

The revision number is wrong. We cannot rely on the comments to tell us the file version. But that's okay because our source control management system gives us the tools to find the right answer. The process is similar for SVN and git.

Tagging a release commit

Tags are useful for marking milestones. Upon release, we can tag the applicable commit with a name based on date like `rel-20230213`. With this information we can checkout this commit or compare using the tag.

If we check out the release tagged commit, all files should match the state of the production environment.

Use history to confirm a change in git

For a given file and a given commit we can see the changes that happened on that commit and compare it with the file under investigation.

Example

Using psidev01 and `dlgGAGateHouseExitNGLO.spb` again, let's examine the timeline of commits for the file.

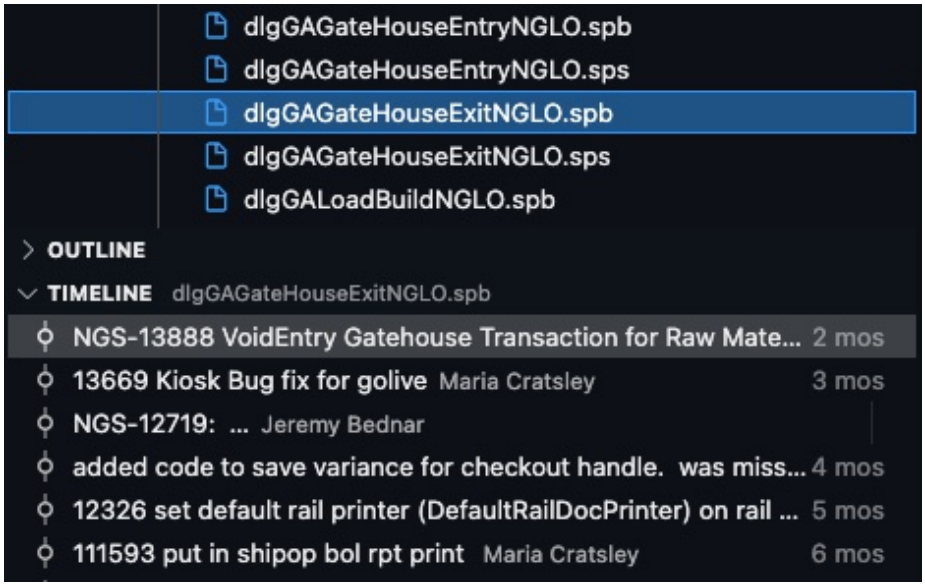

Figure 5: Timeline in VSCode

The VSCode timeline makes it convenient. We can quickly click on each commit and compare the changes with the file in the database.

We can see the last commit (0f608ee9) added a new `checkVoidEntry` procedure starting at line 58. Checking this file in the database (psidev01), this change is not included. We can examine previous commits to find it.

The next previous commit (45ca0b9e) for this file added a conditional block at line 189. That is also not in the database.

The next previous commit (86602618) added a call to `entSHPLoadHistoryNGLO` at line 775. This *is* in the database. We can surmise that this is the commit applied in psidev01.

Update on build, not source control

The last author, last changed date, and commit ID could be added to the database using automated build and deploy tools. They could do it consistently, correctly, without relying on a person to dot hings in a particular sequence. This will be a goal for the near future.

The existing process uses last commit information. It would be trivial for a developer to make change, compile it against the database (with the old revision number), and then commit. The revision number cannot be relied upon, its a guide best which must be corroborated by checking the changes actually exist.

The source control does not account for ad hoc changes that could be applied or migration scripts edited outside of source control.