**Tracking changes made to the SQL Server Database objects (Version Controlling)**

This App is to source control (using GIT), the changes made to any SQL Server database object, by running scripts in management studio. If this is used, there is no need to save any file and push that to the repository and push it to cloud. If it is to be done in the latter method following issues would arise among many.

* Developer forget to save
* Developer save with a wrong file name
* Need to commit and push to GIT manually

When this app is used, no needed to be worried about all the above issues. Especially when someone made a change to a stored procedure and compile (which overwrites the existing SP and there is no way to find out what was there earlier, other than getting the previous backups.

There are few drawbacks in this approach.

* If the changes are made by the SQL Server Management Studio, this ***may*** not work as expected. For ex. If a table is altered, the queries is not “Alter table” as what is really need to be executed.
* If it is needed to record the comment of changing, that has to be done in a particular form, otherwise it cannot be captured.

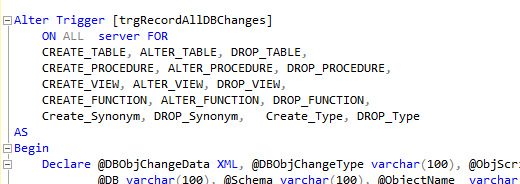
Process has 3 main components

1. Database Components:
   1. A server trigger
   2. A table to record the changes and the GIT repository data.
   3. A GIT repository
2. App which is scheduled to run once a day or week to
   1. Get data (specially the SQL scripts) from table, which are newly added
   2. Save data in a file in the local repository
   3. Commit to the GIT with appropriate comments
   4. Publish the folder (with all changes) in to GIT
   5. Get back the GIT URL
   6. Update the above table-record with the URL and all other data.

**Details**

1.a Server Trigger

Trigger is a DDL trigger, which is running when any DDL change is made to any database object, as Table, Stored Procedure or any such. Any object / changes can be added later by altering the Trigger Script.

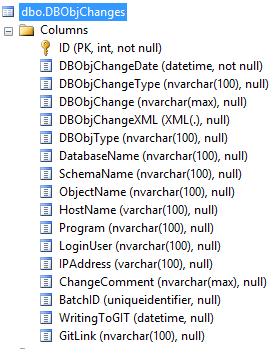


1. b Table to store changes

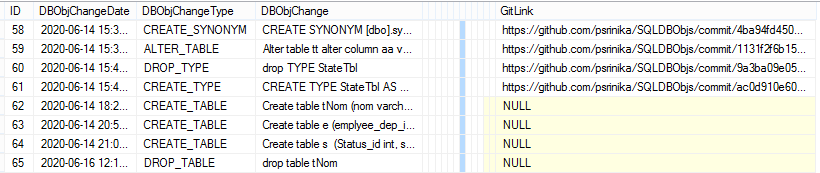
Table is kept in its own database, which has fields to capture the data related to the change and the GIT repository links. Table is filled when any change is made to 4 objects: tables, stored procedures, functions and views. This can be extended to other objects as triggers, synonyms and so on. The trigger needs to be modified in order to capture changes of those objects as well, if needed.

Table is filled by the *Inserts* in the trigger and the table is *updated* when the app run to commit and publish files to GIT repository.

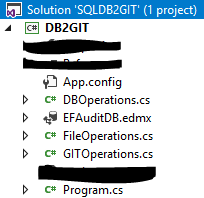
Other than the version control related info, this records the user / IP and such info of the query executed to change the DB Object. This table / DB should be kept accessible by only authorized personnel.

Table Structure  


Data in some of the columns

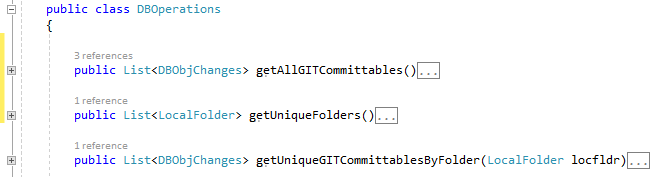


2 App  
  
The App is a .Net (C#) Console App. The app has to be installed in a windows server and the windows / task scheduler has to be attached to this so that this runs in a scheduled time. For a small team weekly running and for a larger team daily scheduling can be done. There is not much impact in the frequency, except that, it is better to schedule to run it off work, to avoid any table lock or such.

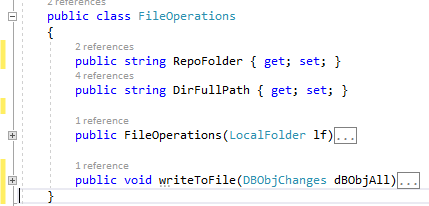


2.a App - Get Data

In the App, EF is used to make data interactions with the DB, where the changes are stored. It has a class, in which there are methods to get, newly added data. Newly added means, the files which are have not been added to the GIT. Those files are found in conjunction with the table which has null values for the batchID (ie. Those records which are not processed – or in other words, the records added after the last GIT update, by running of this app.)  
There are other methods to get each folder name, to get the record details of each data object, etc. changed after the last GIT update.



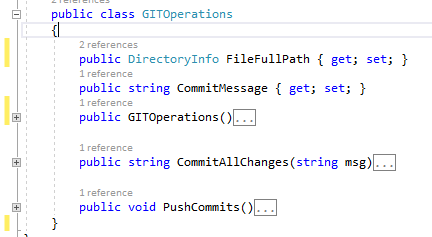
2.b App – Save Changes in to file.

Once the appropriate data is selected above, create the folder (if not exists) and get data from the above 2.a, and save data into a file with appropriate name.  
  


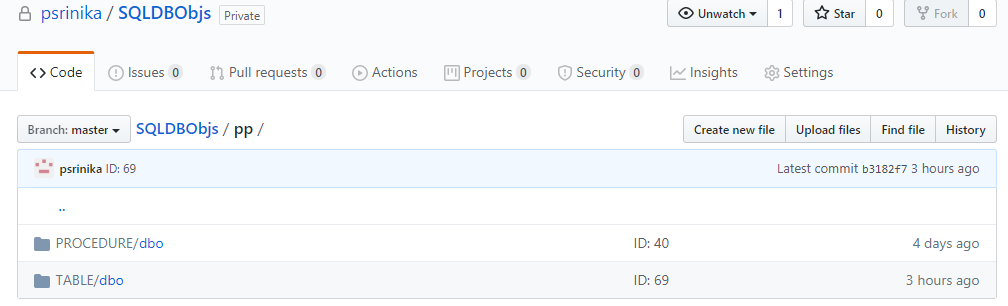
2.c App – GIT commit with comments.  
  
Once all the files of a folder are created, the whole folder is committed to the local repository with comment as the ***ID*** of the record. The comment can be added with the comment in the record, if that exists, but not implemented yet. Cannot set the comment summary and details separately, but both will be the same. The ID is useful if need to trace back to the table, in case need to see the other details.

2.d App – Publish the folder (with all changes) in to GIT

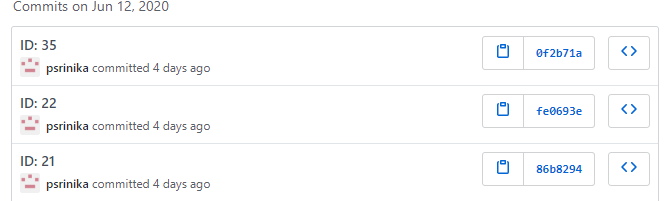
All the commits done in above 2.c are published (synchronised or pushed) to GIT Cloud Repository. All Comments and other details will be published to Cloud.



I used my private repository, but the Repository has to be dedicated to the company which uses it.



2.e App – Get back the GIT URL

When published, each change will have its unique id (a Hexadecimal number) in GIT Cloud Repository. Actually that will be created at the time of commit and that’s why, for each changed file, there is an ID. That id is combined with a fixed URL to reach the change in GIT Cloud Repository. First 7 characters of that id is shown in the page when looked at the commits.  
ex:   
  
 

2.f App – Get back the GIT URL

After getting the Hex ID of the above commit-publish, and concatenating to the fixed portion of URL, the relevant URL is created.  
  
ex.   
Fixed: https://github.com/psrinika/SQLDBObjs/commit/  
Hex ID: ac0d910e601f482e7ac41a915845f66b64e0a3ec  
URL: https://github.com/psrinika/SQLDBObjs/commit/ac0d910e601f482e7ac41a915845f66b64e0a3ec

2.g App – Update the above table-record with the URL and all other data.