Tom2Git

Tom2Git is a tool designed to export **Team Object Manager** (TOM) repositories—an exclusive version management system for Gupta Team Developer projects—into Git repositories.
This tool captures essential check-in information such as user details, timestamps, and labels, transforming them into a structured Git commit history.

The result is a Git repository that mirrors the original TOM repository, preserving its history, changes, and structure in a format compatible with modern version control practices.

Tom2Git is developed using Team Developer 7.5, and the release package includes the required TD runtime.

Why Convert from TOM to Git?

Many organizations are adopting Git as the standard for version control due to its advanced features, widespread adoption, and integration across development tools. Several motivations drive the conversion from TOM to Git:

- **Company Policy**: Organizations standardizing on Git may require all projects to migrate.
- **Unified Version Control**: Teams already using Git may wish to retain TOM repositories for historical or compliance reasons, archiving them in Git to consolidate repositories within a single system.
- **Simplified Maintenance**: Maintaining a TOM database and network storage is often cumbersome. Converting to Git simplifies access, backups, and archiving in a universally supported format.
- **Enhanced Collaboration**: Git provides better support for collaboration, branching, and merging—features that are limited in TOM.
- **Integration with Continuous Workflows**: Git integrates seamlessly with modern CI/CD pipelines and automation tools, enabling continuous integration, testing, and deployment workflows that are challenging to achieve with TOM.

One specific use case is for companies that have already migrated to Git but need to maintain old TOM repositories to access historical source code. Keeping TOM databases running for this purpose can introduce challenges when backend systems (such as databases and storage) become obsolete or need upgrades, risking the loss of valuable project history.

By converting TOM repositories to Git, **Tom2Git** enables easier access to historical data, simplifies storage and maintenance, and aligns with modern development workflows. This transition allows the old TOM repositories to be safely archived or retired.

Features

- **Preserves Historical Data**: Tom2Git captures check-in details (username, timestamp, labels) from TOM and converts them to Git commits, ensuring that the Git history accurately reflects the original TOM history.
- **Automatic Structure Conversion**: Project and file structures from TOM are translated into Git's directory and file structure, creating a fully functional equivalent in Git.
- **Single or Multiple Repository Conversion**: Supports conversion of TOM projects into a single repository (monorepo) or multiple separate repositories (polyrepo).
- **Customizable Commit Information**: Git commit details such as username, email address (for committer and author), and commit messages can be customized to suit your Git repository needs.
- **Snapshots**: Capture snapshots from the TOM repository to reflect the folder and file structure locally based on a specified date or revision within TOM. This feature allows you to limit Git commit history by setting a start point for conversion.

- **Delta Revisions**: Supports files configured as “delta,” ensuring all versions are fully extracted.
- **Compressed Files**: Supports compressed files which will be placed in Git as uncompressed.
- **File Extraction**: Individual file revisions can be extracted from TOM and saved locally.
- **No Git Installation Required**: Tom2Git does not require a local Git installation to create Git repositories.

How It Works

Tom2Git connects to the specified TOM database using the same credentials configured in TOM.
It retrieves all available projects (both active and frozen) along with team member information.
Once you select the projects for conversion and configure Git options, Tom2Git fetches and displays all revisions (check-in information) for the selected projects.

After specifying a work folder to serve as the new Git repository’s location, you can begin the conversion.

The conversion process follows these steps:

- An empty Git repository is initialized in the designated work folder. If in monorepo mode and the work folder already contains a Git repository, the existing repository is opened.
- Optional `.gitattributes` and `.gitignore` files are copied to the repository.
- Each file revision is processed in chronological order, either by extracting it from the database or copying it from storage, depending on the TOM project settings.
- Necessary Git repository subfolders are created based on the structure defined in the TOM project.
- For revisions marked as deltas, the tool reconstructs the full version by combining the master file with its deltas.
- Files are placed in the Git repository, overwriting any existing versions.
- File renames and deletions detected in TOM are reflected in the Git repository as corresponding rename and delete actions.
- New files are added using `git add`.
- Each change is committed to the Git repository using the configured username, email address, timestamp, and a customizable commit message.

These steps are repeated for each file revision in the TOM project until all revisions have been processed and committed.

Once the conversion is complete, the work folder contains a Git repository with the full TOM history represented as individual commits.
This repository is ready for use or archiving.

Setup

Extract the contents of the Tom2Git release package to any location on your system. Ensure all files are present in the folder for the tool to function correctly. Best to place Tom2Git on a location which has full access to the folder contents and subfolders and will not be blocked by Windows or anti-virus tools. The Tom2Git release includes the TD runtime (TD 7.5) along with the main application, `Tom2Git.exe`, allowing you to run the tool without requiring a separate TD IDE or TD runtime installation.

sql.ini

The release package does not include a `sql.ini` file, which is necessary to configure access to the TOM database.
You will need to copy a properly configured `sql.ini` file into the application folder.
Test the `sql.ini` configuration with TOM first. Once verified, the `sql.ini` file can be used with Tom2Git.

The provided TD runtime includes all necessary database-related DLLs to connect to supported TOM databases, such as SqlBase or Oracle.

Usage Guide

Launch Tom2Git and Connect

Run the main application, `Tom2Git.exe`.

Click the “Connect” button in the toolbar to open the login screen, where you should enter the same credentials used for TOM.
Once connected to the TOM database, the main application GUI will be unlocked, and the application will remain connected until you disconnect.

Main Application GUI

The main form consists of two primary sections:

- **Upper Section**: Contains multiple tabs that provide an overview of TOM projects, team members, general Git settings, and the list of revisions to be converted.
- **Lower Section**: Displays details for the selected TOM project, including a list of modules (files) and all checked-in file revisions for that project.

The toolbar includes buttons for connecting/disconnecting, starting exports, and creating snapshots. It also includes settings for selecting the repository type and work folder.

List of TOM Projects

After connecting to the TOM database, the application loads and displays a list of available TOM projects.
Selecting a project row displays details about its modules and file revisions in the lower section of the screen.
Use the checkbox next to each project to indicate whether you want to export it to Git.

You can select multiple projects for export. Depending on the Git destination type (monorepo or polyrepo), each TOM project will be exported either into a single Git repository or into separate repositories for each selected project.

When a project is selected for export, the “Export Revision List” tab shows all file revisions that will be exported for that project. If multiple projects are selected, this tab displays the combined list of file revisions for all selected projects.

Team Members

Tom2Git retrieves and displays a list of all TOM team members in a separate tab. Initially, this tab shows the original team data (code and name) as fetched from the TOM database.

Git requires a name and email address for each commit. In this tab, you can manually provide or update team members’ names and email addresses.

For example, if the names in TOM differ from the names you want to use in Git, you can modify the data here, except for the member code (required to link TOM check-in information to the correct team member).

You can manually enter an email address for each team member or automate this by providing a general email suffix in the “Email Spec.” field. For example:

```
@mycompany.com
```

Once an email specification is entered, the “Names as Email” and “Codes as Email” buttons will automatically populate empty email fields in the member list.

For convenience, you can save this customized team member list using the “Save Template” button. This allows you to reuse the modified list in future Tom2Git sessions

without re-entering data. If needed, you can press “Reset” to discard changes and revert to the original list from TOM.

Git Repository Settings

In this tab, you can specify settings for the Git repository, adjust information used for Git commits, and define the commit granularity.

- **Used Names and Email Information**: The global settings for Author and Committer allow you to define the names and emails used for commits. This ties in with the Team Members section described above. You can choose to use each team member’s name and email from TOM or override it with a single name and email address by selecting “Override Signature.” This option applies the specified global name and email for all commits.

- **Git Handling of Repository Files**: Tom2Git includes two template files to use in newly created Git repositories:

  - **.gitattributes**: Added to the Git repository root, `.gitattributes` controls Git’s handling of file types, text normalization, and end-of-line (EOL) settings, ensuring consistent behavior across platforms. It also allows you to specify custom diff, merge, and export options for specific file types.
  - **.gitignore**: Also placed in the root, `.gitignore` specifies which files and directories Git should ignore, preventing unnecessary files from being tracked and keeping the repository clean.

  Tom2Git stores these template files in the `TEMPLATES` subfolder. You may edit these files before export if you want to customize the settings for your repository. By default, the templates are tailored for TD-related files.

- **Customize Commit Message Text**: By default, Tom2Git uses the following template for commit messages:

  ```
  Tom2Git export: {CURRENTDATETIME}
  TOM project: {DATABASE}/{PROJECTCODE}
  File: {FILE}
  Label: {LABEL}
  Description: {DESCRIPTION}
  ```

  Each commit message is formatted as shown above, with placeholders automatically replaced by actual data. You can modify the template by editing the `TEMPLATES/CommitMessage.txt` file. A list of predefined variables for use in custom messages can be found in `TEMPLATES/CommitMessage_Variables.txt`:

  ```
  {BLOBID}             -> TOM file blob ID
  {CURRENTDATETIME}    -> Current date/time
  {DATABASE}           -> TOM database name at export time
  {DESCRIPTION}        -> Description text of original check-in
  {FILEPATH}           -> File folder + filename
  {FILESIZE}           -> File size (uncompressed)
  {FILE}               -> Filename
  {ISCOMPRESSED}       -> TOM file compression
  {LABEL}              -> Label text of original check-in
  {MODIFIEDBY}         -> TOM member code of check-in
  {MODIFIEDBYNAME}     -> TOM member name of check-in
  {MODIFIEDDATETIME}   -> Original check-in date/time
  {MODULEID}           -> TOM File Module ID
  {ORIGINALFILEDATETIME}-> Original file timestamp
  {PATH}               -> File folder path
  {PROJECTCODE}        -> TOM project code
  {PROJECTNAME}        -> TOM project name
  {PROJECTPATH}        -> TOM project path
  {REVISIONNUMBER}     -> TOM file revision number
  {SOURCEID}           -> TOM file source ID
  {STORAGEMETHOD}      -> TOM file storage method
  ```

- **Branch Name**: Define the branch name to use for the newly created Git repository.

- **Commit Interval**: By default, each TOM check-in corresponds to one Git commit.

Since TOM is file-based, this can result in one commit per modified, added, or deleted file in Git. To reduce the number of commits, you can specify a commit interval to group multiple TOM changes into a single Git commit.

- For example, setting the interval to “1 day” will consolidate all changes within each day into a single commit. This approach reduces the total commit count but also decreases granularity, making it impossible to view specific changes for files checked in on the same day.
- For a more extreme example, setting the interval to “1 year” will group all changes for a given year into one commit, resulting in significant history loss.

Remember, setting “No Interval” will preserve all changes, creating a complete history but generating more commits. Adjust this setting based on the level of detail you want to retain.

- **Pause After Git Init**: By default, Tom2Git initializes the work folder as a new Git repository and then processes TOM revisions in chronological order. The “Pause After Git INIT” option halts the export just after initializing the Git repository, allowing you to add extra files to the work folder (such as documentation or resources) before continuing. Dismissing the pause message box will resume the export, and the additional files will be included in the first commit.

Export Revision List / Snapshots

The complete list of TOM revisions to be exported (based on the selected TOM projects) is displayed in tab “Export Revision List”. It shows in chronological order all TOM revisions to be exported and committed.

Snapshot You can choose to export the entire list or set a specific starting point, known as a snapshot.

A snapshot captures the state of all files and folder structures within the TOM project as of a specified date or revision. When a snapshot is set, Tom2Git reconstructs the workfolder to mirror the project’s exact state at that point in time, bypassing all commits up to that moment. After this, Tom2Git will begin committing changes to the Git repository from the snapshot onward.

This feature is useful for reducing the commit history by limiting the number of commits included.

Example:
Suppose the TOM project contains a decade of application history, with check-in information for versions v1.0 through v7.0. You may not need the full history and instead want Git to track changes starting from a specific version, such as v5.0 through v7.0.

By setting a snapshot at the first check-in for version 5.0 in the TOM revision history, Tom2Git will reconstruct the complete folder structure and files up to that point without committing them. The first Git commit will occur from the snapshot date, beginning the Git history at the state of the TOM project at the start of version 5.0.

This approach can significantly reduce Git history if TOM projects have accumulated extensive check-ins over many years, allowing you to start the repository at the desired TOM project state.

Set snapshot In the list of revisions to be exported, locate the point where you want to start committing to the Git repository. Right-click on the row for this first commit, and select from the context menu:

- **Snapshot/Set snapshot**
- **Snapshot/Clear**

Once a snapshot is set, all prior revisions will appear in gray, indicating they will not be committed. During export, the first commit will begin from this snapshot revision.

If multiple TOM projects are selected, Tom2Git will automatically align the snapshot across all projects, starting commits based on the snapshot’s date and time for each

project. For instance, if one project has fewer revisions than another, Tom2Git will ensure the snapshot represents the synchronized state across all projects at that moment.

TOM project modules and revisions

The lower section of the main screen displays details about the modules (files) and revisions for the selected TOM project above. Here, you can view more details for each revision, extract revisions, and set a snapshot specific to that project.

Revision Details
Selecting a revision and clicking “Details” opens a popup window with information such as the revision number, TOM database IDs, label text, description, and indicators for whether the file is a delta or compressed. This window can remain open, automatically updating as you select different revisions in the list.

Extract Revision
To extract and save a revision locally for review, select the revision and use the “(Save) Revision” button or right-click and choose “Extract revision” from the context menu. A file dialog will open, allowing you to specify the save location on your system.

Set Project Snapshot
This snapshot feature operates at the individual project level, allowing you to set a unique snapshot for each project’s revisions. Tom2Git will use this exact snapshot point, enabling you to create specific snapshot moments independently for each TOM project.

Git destination (poly vs mono repo) considerations

The **Destination** toolbar option controls how Tom2Git exports your selected TOM projects:

- **Mono repo**: Exports all selected TOM projects into a single Git repository, combining their history and structure into one unified repository.
- **Poly repo**: Exports each selected TOM project as a separate Git repository, preserving each project’s history independently.

If the TOM repository includes related projects that should be represented together in Git, select the mono repo option. If the projects are unrelated in terms of files and history, select the poly repo option.

Example:

TOM Project Code	TOM Project Name
APPSRC	MyApplicationSource
APPDOCS	MyApplicationDocs
APPBMP	MyApplicationBitmaps
TOOLS	General Tools
APPSRC2	AnotherApplicationSource
LICTOOL	LicenseTool

In this case, the first three projects are related but exist as separate projects in TOM. These would ideally be combined into one Git repository. The “General Tools” project, containing demo sources and third-party tools, may be better suited as a separate repository due to its unrelated content. Similarly, “AnotherApplicationSource” could be exported as its own repository.

Access control is another consideration. For example, the “LicenseTool” project might have restricted access in TOM and should be a standalone Git repository if user access needs to remain limited.

Based on these considerations, you may wish to export TOM projects in separate sessions: first exporting mono repositories and then poly repositories as needed. A single export session can only handle either mono or poly repositories at a time.

Global root mode (subfolder name) considerations

The structure of TOM projects should be reflected in Git as well. Since Git organizes files by folder structure, it's essential to determine how the TOM project level will be represented.

By default, Tom2Git uses the TOM project code as the root folder name. For instance, if the first three projects mentioned above in the example are configured as a mono repo, the structure would appear as:

```
WorkFolder\APPSRC\
WorkFolder\APPDOCS\
WorkFolder\APPBMP\
```

Here, the **WorkFolder** serves as the root of the Git repository, containing a subfolder for each TOM project.

This default behavior can be customized by selecting a different root mode.

On the **Tom Repository** tab, the **Global root mode** combo box provides these options:

- **Project code**: Uses the TOM project code as the subfolder name.
- **Project name**: Uses the TOM project name as the subfolder name.
- **Custom name**: Allows you to specify a custom subfolder name.

If "Custom name" is selected, the **Root folder** column in the TOM projects table becomes editable, allowing you to enter a custom name.

For poly repos, each subfolder becomes the root of its own Git repository. In this case, the **WorkFolder** itself is not a Git repository but contains subfolders that each serve as the root Git folder for individual repositories.

Exporting

After configuring all settings, click **Export** to begin processing the selected TOM projects. Tom2Git will validate that the export is ready by checking for required information (such as team member settings) and confirming the workfolder's state. If validation fails, a message box will display the identified issues. Export can proceed only after validation is successful.

When the given Workfolder already contains files and subfolders the validation will fail in the case of polyrepo. When using monorepo and the Workfolder is already am existing Git repository, the validation will succeed. Only monorepo exports are able to be added to existing Git repo's.

Following validation, an overview screen appears, showing the selected TOM projects and any set snapshots, which can still be adjusted at this stage.

Click **Start Export** to initiate the actual export process. During export, a **CANCEL** button is available to abort if needed.

Tom2Git generates a log file detailing each action performed. Any errors encountered are recorded in the log for troubleshooting purposes. The log file can be accessed through the Tools menu or the provided buttons on the toolbar.

Once complete, a confirmation message box will display the export results. For successful exports, the working folder will contain either a mono repository or multiple subfolders as separate Git repositories for poly repos.

Snapshots

As described, snapshots capture specific points in time for TOM revisions and serve as the starting point for commits during export to Git repositories.

Additionally, the toolbar includes a **Create Snapshot** button (located next to **Export**), which operates differently. This feature creates a snapshot without initiating a Git

repository. Instead, it recreates the files and folder structure in the working folder as they existed at the snapshot point, simulating a manual extraction of all files and subfolders from TOM for that date or revision.

For example, if you need the TOM project files as they were at the release of version 5.0 (from five years ago), set a snapshot on the final check-in for version 5.0, and execute the snapshot. The working folder will contain the files in their exact state at that historical moment.

This method is efficient since it bypasses Git repository creation and avoids performing `git add` and `git commit` operations, simply extracting files to the workfolder without the overhead of Git.

This feature is especially useful when still working with TOM, even if there is no intent to convert to Git, allowing you to recreate TOM history up to a specified revision and extract relevant files—an option that TOM itself does not provide.

Tools menu

The tools menu of the main screen has several helper options like opening work and temp folders.
Also **TDAppTools** is included which makes it possible to debug database access (sql statements), inspect GUI handling and gives detailed info on the application itself.

Tom2Git.cfg

When using Tom2Git, many settings within the application are saved to `Tom2Git.cfg` file, next to the main executable. Subsequent running of Tom2Git will load the previously set options. To reset the options, delete the file.

Troubleshooting, limitations and requirements

- **Connection Fails**: Ensure you have the necessary permissions to access the TOM database and that your system meets all prerequisites. First, test TOM on the same system where Tom2Git will run, using the same `sql.ini` configuration for Tom2Git to connect correctly.
- **Export Errors**: Export issues often arise due to corrupted TOM files in the database or missing file storage. Check TOM and attempt to extract any problematic files directly. If TOM cannot extract these files, Tom2Git will encounter the same issue.
- **Abort on Error**: If files are corrupted or unavailable during export, you can uncheck the “Abort on error” option to continue the export despite these errors. This may result in missing revisions, but the rest will be converted. You can attempt to resolve the issue directly within TOM (using access to the TOM database or file storage). Additionally, you have the option to manually address file issues after export and make manual Git commits to correct any missing or corrupted files.
- **Data Verification**: After export, carefully review the Git commit history to confirm that all check-in data (user, date, labels) has been accurately converted. Tom2Git does not guarantee complete accuracy, so thorough verification is essential before deprecating TOM. It is the user’s responsibility to ensure the export produced a complete and usable Git repository.
- **Single/Multi-user database**: When TOM is used on a single user database (like single-user SqlBase provided with TD), be sure not to run TOM and Tom2Git at the same time. Tom2Git will connect and stay connected to the database which may block TOM to be started and visa versa. Also make sure that no changes are performed in the TOM database during export and usage of Tom2Git. This may lead to issues.
- **Read only access**: Tom2Git only reads data from the TOM database and does not change any data. It may block actions from other TOM instances used on the database so be sure to run Tom2Git only when no other users are connected.

Tom2Git project and development

Tom2Git is developed using the latest version of Team Developer, currently TD 7.5. The project root folder includes the following files:

- **TomToGit.apt**: Main source file in text format (TD 7.5).
- **LibGit2TD**: Reference libraries (APL, DLL) for managing Git repositories.
- **TDAppTools**: Debugging plugin for applications (includes SQL monitor, GUI inspector, etc.).
- **Callback**: Helper library for advanced Windows callback implementations.
- **TmDfi75.dll**: Original TOM runtime library to support delta file revisions.
- **OpenWith_TD75.bat**: Batch file to open TD sources in TD 7.5. Drag and drop the main source file onto this batch file in Windows Explorer to open it in the TD 7.5 IDE.
- **CreateRelease.bat**: Builds Tom2Git and creates a "Release" folder containing all runtime files, suitable for official releases.

If you encounter issues, you can troubleshoot using the provided project files. Various factors (such as database brand, file storage, or Windows versions) can impact functionality, and having the complete project files available may help resolve these issues.

System Compatibility:
Tom2Git has been tested on a limited range of systems with different TOM databases and project settings. If you encounter issues with your setup, please provide feedback to help improve Tom2Git's stability and compatibility across more environments.

Contributing

If you find this project helpful and would like to improve it, contributions are very welcome. Any assistance, including pull requests, is appreciated.
This is initially started as a personal project which will be quite useful for the TD community.

If you'd like to become an official contributor, please contact me to be added to the project.

TD Community Forum

Join the TD Community Forum for questions, answers, and discussions related to Gupta Team Developer:

https://forum.tdcommunity.net

You can find me on the forum under the name "Dave Rabelink".