**HOW TO Convert a Subversion (SVN) Repository to a JPL GitHub Repository**

Note: This procedure assumes you are working on a Windows (7/8/10) system.

1. **Converting a Subversion Repository to a Local Git Repository**
2. Install needed applications:
   1. Install Notepad++: <https://notepad-plus-plus.org/download/>
   2. Install Git for Windows: <https://git-scm.com/download/win>
      * Follow installation hints on the TortoiseGit “FAQ: System prerequisites and installation” page, <https://tortoisegit.org/support/faq/#prerequisites>
   3. Install TortoiseGit: <https://tortoisegit.org/download>
      * Select all the defaults, but recommend setting the editor to Notepad++
   4. Install GitHub Desktop: <https://desktop.github.com/>
   5. Install GitHub’s Atom editor: <https://atom.io/> (recommended)
3. Obtain the URL to the Subversion repository you will clone & convert.
   1. If this is a repo in our SVN server, you can open a web browser and surf to the UberSVN portal, <https://msi-server.jpl.nasa.gov:9900/>, to get the URL.
   2. Log into the UberSVN portal using your JPL credentials.
   3. Click the "Repositories" tab, then surf to find the repo you wish to convert.
   4. Swipe and copy the corresponding URL in the center column.
4. Open the Windows Explorer file browser and surf to the folder that contains (or will contain) your collection of Git working copy folders.
   1. E.g., 'cd C:\users\<username>\documents\github'.

1. In the right side pane that displays this folder's contents, right-click in a blank area and use the TortoiseGit menu to select "Git Clone..."
   1. The "Git Clone" dialog box should open, and should automatically paste the copied SVN repository URL into the "URL:" textbox for you.
      * Note that you can set/change the URL for the source repository by clicking the "Browse..." button to the right of the "URL:" textbox, but this will only work for URLs of type 'file://' that point to SVN/Git repos that are located locally (i.e., on your system); it won't allow you surf to another machine this way.
      * TortoiseGit can use 'http', 'https', 'git', and 'ssh' protocols to reach other machines on the network, but you will have to specify them here by typing out their full URL.
   2. TortoiseGit should also automatically set the "Directory:" textbox to the file path for your Git "working copies" base folder, appending a subfolder with a suggested name matching that of the repository to be converted.
      * You can edit the destination folder name as you wish, or you can change its name & location by clicking the "Browse..." button to the right of the "Directory:" textbox and surfing to/creating a new destination folder.
   3. Check the "From SVN Repository" checkbox. This disables settings for cloning another Git repository and enables the settings for cloning an SVN repository.
   4. You will normally want to leave the "Trunk:", "Tags:", and "Branch:" checkboxes checked, and their names unchanged. However, if you have used non-standard names or folder structures in your SVN repository, you may need to change these fields accordingly.
   5. Leave the "From:" checkbox unchecked to convert the entire SVN history to Git (which is likely what you want to do). Otherwise, check this checkbox and enter the number of the starting SVN revision that TortoiseGit should use to begin conversion.
   6. Check the "Username:" checkbox and enter the username in the textbox that corresponds to the SVN server account. TortoiseGit will use this username to log into the SVN server to retrieve the repo history.
   7. Click OK. When using the MSI Server Subversion repository for the first time, you can expect to get a dialog box regarding a "certificate problem". If so, enter "p" in the textbox and click OK to continue. (This is due to the self-signed TLS certificate used by the UberSVN Apache webserver.)
   8. The next dialog box that appears should ask you for a password. This is the password for the SVN server account corresponding to the username you entered in the "Git Clone" dialog box.
   9. Click OK to begin the conversion. A dialog box showing the repo conversion log should appear. You can resize this dialog to see more of the log details as conversion progresses. Note that it may take several seconds for TortoiseGit to connect to the SVN repo and begin logging revision conversions one by one. It will progress from the initial rev to the HEAD rev, and notify you in the log display when conversion is complete.
   10. Once the process has completed, you can close the conversion dialog. The converted Git repo will be in a folder named & located as specified in the "Git Clone" dialog. This folder should contain both a '.git' repo folder and a full working copy of the HEAD revision.
   11. Note that if the SVN repository has branches and tags, the SVN branches and tags folders are NOT in the local repo! You have to fetch them to the local repo if you want them pushed to the new GitHub repo. Since you're migrating from SVN to GitHub, you \*want\* to do this, otherwise they’ll be lost. For this step, I found using 'GitHub Desktop' works well (since it's point-and-click, so it's quick).
   12. In ‘GitHub Desktop’, add the converted local repo in that you just converted, then pull down the “Branches” control to select each branch and each tag, one by one. You’ll notice that it takes some time to switch to each; GitHub Desktop is fetching them from the remote SVN repo to the local Git repo. Once you've fetched all the branches and tags, go back to the “master” branch.
2. In the Windows Explorer file browser, right-click in a blank area in the working copy and use the TortoiseGit menus to select "Git > Settings".
   1. In the "Git" pane, in the "User Info" block, set your username and email address. Git uses this username to label your Git activities in the log, etc.
   2. You may also wish to configure additional settings at this time, such as Context Menu items (e.g., Sync, Commit, Show log, Repo-browser, Rename, Delete), Icon Overlays, and Diff Viewer apps.
      * E.g., to support LabVIEW development, click 'Advanced' to set entries that call the LabVIEW “Compare VI” app. (See the HOW-TO document on this subject for command wording and other details.)
3. At this point, you can start editing the working copy files; add/delete files, etc., in the working copy; and commit to the local Git repo. You can use the command line, GitHub Desktop, or the TortoiseGit menu in Windows Explorer to do this.
   1. For example, use the "Git Commit ->" menu item after making changes to the working copy files to commit your changes to your local Git repo.
4. This initial conversion produces a local working copy of the project, along with a Git repository (in the accompanying '.git' folder) that contains the entire commit history. This may be sufficient in some cases, but in general you will want to “reverse clone’ the repo to a remote server such as GitHub.
   1. An upstream repository allows you to access your repo from another machine, enables sharing and collaboration, and provides a convenient means of backup.
   2. Shared Git repos are "bare" repos (meaning, they have no working copy files associated with them). Cloning from, and pushing upstream to a non-bare repo risks causing inconsistencies when files in a push conflict with working copy files.
   3. The preferred solution is to export your repo to a remote server, then set that as your upstream (typically referred to as 'origin'). Other team members can then clone that repo to their systems as their origin and begin collaboration.
5. **Importing a Local Git Repository to GitHub**
6. Log into the JPL GitHub server, <https://github.jpl.nasa.gov/>, using your JPL credentials.
7. If you have already created an (***empty***) repository for this project in GitHub, then surf to its "Code" tab and click the "Clone or download" (green) button.
   1. However, if you are creating the repository at this point, click the '+' menu in the upper right corner (next to your avatar icon) and select "New repository".
   2. Fill in the "Repository name" and "Description" boxes as appropriate.
   3. Select "Public" or "Private" for the repository. (Private repositories can have collaborators.)
   4. Do *not* check "Initialize this repository with a README" (since you will be importing a repository from your system, and the repo needs to be *empty*).
   5. Click "Create repository". This will bring up a new web page...
8. (While this next step can be done entirely within the TortoiseGit dialogs, it’s somewhat easier and more straightforward to use the command line.) Look in your Windows Start Menu under 'Git' and open 'Git CMD' ('git-cmd.exe').
   1. Use the command line to surf to the working copy folder for the converted repository. (E.g., "cd /users/<username>/documents/github".)
   2. Type "git remote add origin " (with the extra space) -- but don't press <Enter> yet. (Note that you can swipe & copy the text in this procedure and right-click in the 'Git CMD' window to paste it).
   3. Return to the GitHub web page; find and click the 'copy to clipboard' icon on the far right side of the "Clone with HTTPS" or "Quick setup" box (depending on whether or not you just created the repository) to copy the repo URL to the clipboard.
   4. Right-click in the 'Git CMD' window to paste the URL, then press <Enter> to enter the command. This will add your GitHub repo to your local repo as a remote repository named "origin".
   5. Enter "git push --all -u origin" (i.e., swipe this command, right-click & select “copy”, then right-click in the 'Git CMD' window to paste it) to push all branches & tags in the local repo to the GitHub repo. The '-u' switch in this command will set the GitHub repo as the upstream "origin" for your local working copy.
      * If the remote repository on GitHub was not empty, you may get an error here regarding “unrelated histories”, with Git refusing to merge the repos. It is possible to do a "git merge" by hand using the "--allow-unrelated-histories" option; refer to Git documentation for details.
   6. You no longer need the 'Git CMD' window (unless you prefer using Git on the command line; if you do, you also have the option of using the ‘Git Bash’ shell (if you prefer Linux syntax to Windows syntax).
9. Refresh the repository 'Code' page in the web browser. You should see line items for the commit history for your project, confirming that it has been imported.
10. From this point on, you can use the command line, TortoiseGit, and/or GitHub Desktop to work with your repository/working copy, as you prefer.
    1. If you prefer to use TortoiseGit, its major difference from TortoiseSVN is an additional dialog used to push/pull (sync) a working copy’s repo with its upstream on the GitHub server. Right-click in your working copy and use the TortoiseGit menu to select "Git Sync..." to open this "Git Synchronization" dialog.