

CVS Migrations Overview

When looking to migrate your existing CVS repository to Subversion using `cvs2svn.py`, there are a number of approaches available to you in migrating that data, depending on your needs. Your choice will be dependent upon how much historical data you want in your Subversion repository. It may be satisfactory to just refer to the existing CVS repository for the historical data created when it was the production system and start work in the new Subversion repository with just the latest revision. Alternatively, you may want to move all the history that's accumulated in your CVS repository into the new Subversion repository. Somewhere in between these two approaches are other types of conversions with some, but not all, data migrated from your CVS repository into Subversion. This document tries to describe common paths to accomplishing whichever migration approach best meets your project's needs as well as highlighting the advantages and disadvantages of each approach..

Tip or Top-skim (hint: you don't need `cvs2svn` for this one)

This approach is the quickest and easiest way to get started with your new Subversion repository. Basically, you are exporting the latest revision (or whatever versions you want to start with) of your CVS repository and then importing it into your Subversion repository. Keep in mind that you can also take this opportunity to refactor (i.e., reorganize) the directories and files before you execute the import and establish a baseline structure within your Subversion repository.

As a common best practice, this means that you will also keep your CVS repository available as a read-only reference when history needs to be reviewed. That may mean that you keep it available on a server somewhere or that you archive the repository (and CVS) so it is available, but requires a bit of work to restore. If you determine that you do not need or want the historical data, you can of course just delete it and ignore the fact that anything happened historically before Subversion entered your life.

Advantages:

- *Clean* - A concise starting point is established while excluding the clutter found in the legacy repository. The repository starts with a minimum size allowing the disk space to be used for future development.
- *Quick* - A single command line operation imports the data and gets the project team up and running with limited impact and in the briefest amount of time.
- *Convenient* - The project team can easily execute this operation without any need for additional resources and therefore, at exactly the time best suited for that project.

Disadvantage:

- *History* - Any need to access historical data will require the user to access the CVS repository (which may also require that the repository be restored from an archive).

Trunk Only

It may be that you primarily develop off the trunk in your CVS repository or that you use the trunk as your milestone release branch. In cases like these, you may just want to migrate just the trunk to Subversion as other branches and tags are not that useful. The cvs2svn script provides a single switch to accommodate this approach - `--trunk-only`.

Advantages:

- *Clean* - While not as compact as the Tip migration, this approach still focuses on a limited set of revisions while pruning the remainder of the version tree.
- *Quick* - Again, this approach will not be as quick as the tip migration, but the pruning of additional branches and tags can have a significant impact on the time required to execute the migration versus a full migration.

Disadvantages:

- *Duration* - While the duration will vary greatly based on the amount of revisions created on the trunk, it will, by nature, be a considerably longer migration process than the tip approach above.
- *History* - Any need to access tags or revisions on branches will require the user to access the CVS repository (which may also require that the repository be restored from an archive).

Selective Migration

If your CVS repository has daily build tags and developer branches that you do not want migrated into your Subversion repository, but it has other tags and branches that you do want, then a selective migration is what you need to select. In order to execute this approach, you'll use the `--exclude` switch to inform cvs2svn which tags and branches should be ignored during the migration.

Advantages:

- *Concise* - Less compact than the previous approaches, this still focuses on a limited set of revisions that are perceived to be of the most value to you while pruning the remainder of the version tree.

- *Quick* - Again, this approach will not be as quick as the tip migration, but the pruning of additional branches and tags can have a significant impact on the time required to execute the migration versus a full migration.

Disadvantages:

- *Duration* - While the duration will vary greatly based on the amount of revisions created on the branches and tags you include, it will, by nature, be a considerably longer migration process than the tip approach above.
- *History* - Any need to access tags or revisions on branches that were not migrated will require the user to access the CVS repository (which may also require that the repository be restored from an archive).

Full Migration

The obvious gut reaction to what needs to be migrated is always everything. Interestingly enough, this is the default behavior for cvs2svn so it is straightforward to execute. Keep in mind that rarely is it the case that all data should be migrated or will bring value after being migrated.

Advantage:

- *History* - Everything is available from the Subversion repository with no need to support a CVS repository or archive it. Any access of history will be convenient for the user.

Disadvantages:

- *Duration* - Duration will vary greatly based on the amount of revisions, branches, tags, and potential atomic commits. It can be the longest migration approach.
- *Clutter* - Every mistake every committed, every intermediary commit (however pointless), every developer and feature branch, every meaningless tag, etc. will be cluttering the Subversion repository as it did the CVS one. Branches and tags will likely require additional setup work to reorganize them so the user is not overwhelmed with all of them appearing at the root level of the Subversion '/branches' directory.

Combinations

There will be occasions when none of the previous approaches will uniquely implement the migration that you want for your project, but executing a combination of these approaches can create the repository with the appropriate data that you want. This approach will take a bit more planning, and more time in execution, but you should be able to create a repository with the appropriate data to meet your specific needs.

Advantage:

- *Choice* - This approach gives you the ultimate customization ability for the repository. You can create a migration that is as unique as the data in your repository.

Disadvantages:

- *Duration* - Duration will vary greatly based on the amount of revisions, branches, tags, and potential atomic commits. It can be the longest migration approach depending on the complexities of the actual migration steps required.
- *Complicated* - Freedom to define a unique migration process has the obvious tradeoff that the process will be more complicated, time-consuming, and error-prone than other approaches.
- *Broken* – This approach is likely to break commands that use date-based ranges since Subversion does a binary search through the repository for dates. This is not a huge disadvantage, but certainly qualifies as an annoyance.

Single Project

All the above approaches assumed an overall approach to an entire repository, but you may have many different projects that you cannot or do not want to migrate simultaneously. For example, you may want to create separate repositories for individual projects in Subversion. This approach will require multiple migration phases, but in breaking down a large CVS repository, it will make the effort much easier to execute in a timely manner for each individual project. This approach is accomplished by providing the path to the project rather than the root of the repository when executing any of the previous methods.

Advantages:

- *Duration* - An individual project will get migrated in less time than if it depended on the whole repository being migrated.
- *Choice* - This approach gives you the ability to be more responsive to individual projects both in the timing of their migration and the potential of isolating them to their own Subversion repository..

Disadvantages:

- *Duration* - Duration is still dependent upon the size of the repository, but doing the work a project at a time will clearly take longer as a whole.
- *Broken* – For cases where this approach is used to migrate to the same Subversion repository, but at multiple point, this approach is likely to break commands that use date-based ranges since Subversion does a binary search through the repository for dates. This is not a huge disadvantage, but certainly qualifies as an annoyance.

Whichever approach you decide to use to migrate your CVS repository, you should use the opportunity to perform maintenance on your data. Directories and files that are not useful should be removed either before or after the migration procedures. And, as mentioned earlier, any refactoring (i.e., reorganizing) of the directories and files should also be done now. This is a unique opportunity to create a new starting point for your repository when all the users are already embracing a change in the version control tool and in the associated configuration management processes.