



Migrate2SVN

User Guide

Table of Contents

- 1 Introduction
 - 1.1 Overview of Migration with Migrate2SVN
- 2 Using Migrate2SVN
 - 2.1 Migration
 - 2.1.1 Simple Migration
 - 2.1.2 Migrate Development Stream followed by Full history
 - 2.1.2.1 Example migration of Development stream
 - 2.2 Migrating Labels to Tags
 - 2.3 Migrating Branches
 - 2.4 Applying Subversion Properties During Migration
- 3 Using the Data After Migration
 - 3.1 Migrated Branches
 - 3.2 Migrated Tags
 - 3.3 Retrieving Log Information
 - 3.4 Working with Migrated Branches
- 4 Migrating to Git
 - 4.1 Subversion Repository Migration
 - 4.2 Why Are Tags and Branches Not Migrated to Git
- 5 Migrating to Mercurial
 - 5.1 Subversion Migration

Introduction

Migrate2SVN allows you to migrate a ClearCase database into a Subversion repository whilst retaining full metadata including authors, timestamps and checkin comments.

Migrate2SVN can also migrate branches and labels into Subversion branches and tags.

Overview of Migration with Migrate2SVN

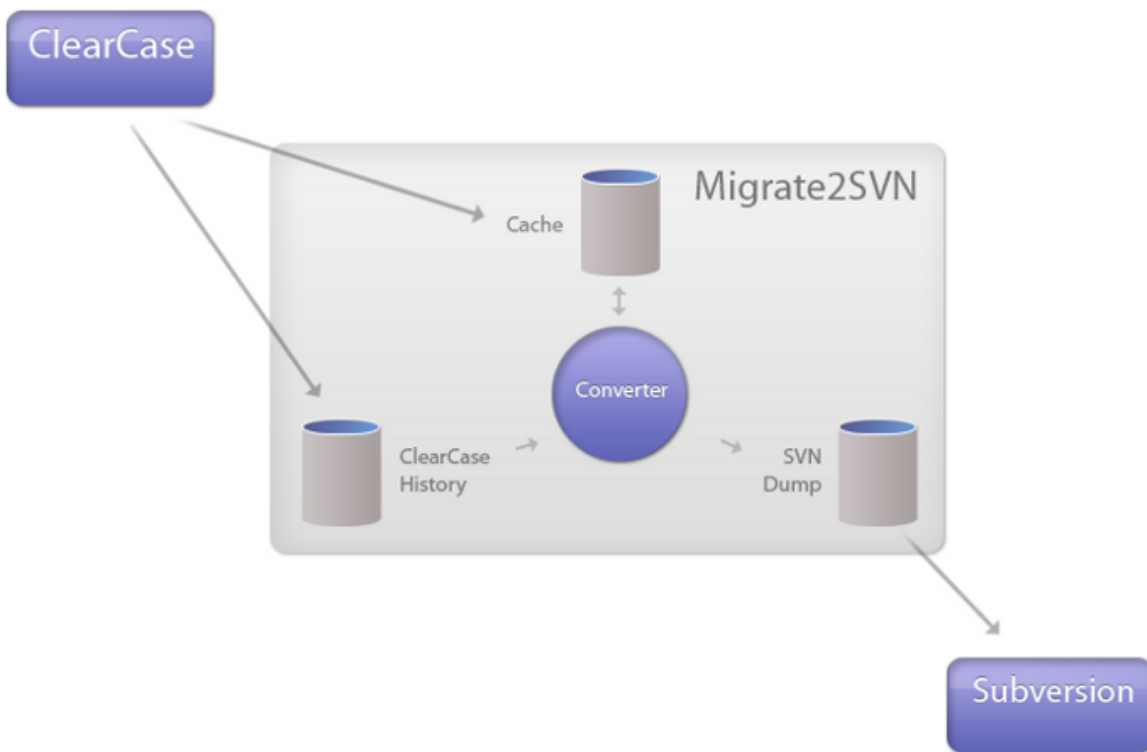
Migrate2SVN operates within the context of a ClearCase view and extracts the ClearCase history for all files and directories contained within the view. It converts the information extracted into a Subversion dump file, which can be loaded into a Subversion repository using standard Subversion commands.

Every new file version in ClearCase is converted into a new Subversion revision, retaining the original checkin's author, timestamp and checkin comment.

Migrate2SVN will also detect labels on ClearCase elements in order to build up tags within the Subversion tags area. In order to ensure that tags are migrated fully, including any elements currently eclipsed by the ClearCase view used for the migration, you can specify a number of config specs to be used for the full migration of labels to Subversion.

When migrating repositories, Migrate2SVN will first extract the ClearCase history to a file and process the file line by line. For each element version to be transferred, Migrate2SVN will fetch the version into a temporary staging area (cache).

This cache allows you to reuse the data fetched during a previous run. The main advantage of this approach lies within the ability to perform a test migration, which will populate the staging area, followed by the actual migration, which will reuse the elements already fetched and only fetch any new elements created since the test migration, speeding up the migration process.



Using Migrate2SVN

When migrating data from ClearCase to Subversion with Migrate2SVN, you need to decide on your requirements with respect to the branches and tags to be created.

You are advised to identify all branches and tags you wish to migrate up-front and provide a ClearCase config spec for each branch and tag, which gives you full control over how tags and branches are created. What you see for a branch in a ClearCase view is controlled by the config spec, whereas in Subversion a branch by convention stretches across a complete repository.

Migration

Simple Migration

Migrate2SVN is a command-line tool, which should be run from within the installation directory. It is therefore recommended that you change your current working directory to the installation directory and run the tool from there.

Linux:

```
cd /opt/Clearvision/migrate2svn/  
./migrate2svn --help
```

Windows:

```
cd c:\Program Files\Clearvision\migrate2svn  
migrate2svn --help
```

The minimum steps required to perform a migration are as follows. Edit the Migrate2SVN configuration file and point the tool to a suitable ClearCase view. The `migrate2svn.cfg` file lives in the same directory as the Migrate2SVN tool.

Note that you can maintain multiple config files and point Migrate2SVN to the config file you would like to use for a particular run using the `-f` option, e.g.:


```
migrate2svn -f another_config.cfg
```

To start a simple migration, edit the config file and configure the view-extended path to a VOB via a ClearCase view.

```
cc_view_dir=m:\myview\myvob
```

This simple migration assumes that you have `cleartool` in your path and that the default settings in the config file are suitable.

Please refer to the Migrate2SVN Administration Guide for full details on all configuration options. However, please note that the default configuration file is annotated with comments describing the meaning of options.

 If you are migrating a new VOB and you have previously used Migrate2SVN for another VOB, you must make sure that the cache area (as defined through the `cache_dir` config option) is deleted prior to this. You must never use the same cache for two different VOBs.

Now run the `migrate2svn` tool and perform the migration.

```
migrate2svn --run
```

When the migration is finished, you will find two new files in the current working directory: `cchistory.txt` and `svndump.txt`. The file that is of most interest to you is `svndump.txt`, which contains the data for the Subversion repository to be created as a Subversion dump file.

All that remains to be done now is to create a new, empty Subversion repository and import the Subversion dump file into it.

On Unix:

```
svnadmin create svnrepo  
cat svndump.txt | svnadmin load svnrepo
```

On Windows:

```
svnadmin create svnrepo
type svndump.txt | svnadmin load svnrepo
```

Migrate Development Stream followed by Full history

Migrate2SVN allows migration of the full history from your ClearCase VOB into Subversion. However, VOBs with large amounts of history may take a long time to migrate and not be practical in your working process. If you believe that it is not realistic to run a FULL migration all at once, it is possible to migrate only the files required to carry on development within Subversion before migrating the full history found within your VOB. Below is an example of how this can be performed (the company used within the example is purely fictional)

Example migration of Development stream

GATE are a mobile phone development company who have a number of VOBs with each having a large amount of history. GATE want to get their developers up and running within Subversion as soon as possible. To do this, they want to migrate the bare minimum of files required for their developers and then run a full history migration afterwards. Their development VOB, called *dev_vob*, takes the form of the structure found below:

- dev
 - screen
 - dev_files
 - java
 - C++
 - old_unused_dev_files
 - keypad
 - dev_files
 - java
 - C++
 - old_unused_dev_file
 - battery
 - dev_files
 - java
 - C++
 - old_unused_dev_files
 - os
 - dev_files
 - java
 - C++
 - old_unused_dev_files
 - media
 - audio
 - old_unused_audio_files
 - graphics
 - old_unused_graphics_files

This is their structure for *dev_vob*. There are numerous files and folders which are not required for immediate continued development within Subversion. In essence with trying to limit downtime, these folders are not required so will not be initially migrated to Subversion. **ONLY** the folders and files which are required for the developers to be able to continue development will be migrated using Subversion commands. GATE have found that only the following folders are required to allow their developers to continue work:

- dev
 - screen
 - dev_files
 - java
 - keypad
 - dev_files
 - java
 - battery
 - dev_files
 - java
 - os
 - dev_files
 - java
 - media
 - audio
 - graphics

It is important to note that Migrate2SVN will not be used to perform the initial development stream migration. Commands provided by ClearCase and Subversion will be used for the initial import and Migrate2SVN will be used to perform the full history migration.

dev_vob is located at `/views/GATE_view/GATE_vob/dev_vob`

After the required folders for the development stream migration have been located, apply a new ClearCase label to each folder. GATE chose to use the label name 'DEVSTREAM' for each folder so they are easily identifiable. GATE then applied a custom CONFIG SPEC to *dev_vob* which only showed files which have been labeled with DEVSTREAM. The config spec can be found below:

```
element * DEVSTREAM
```

The reason for this is that now, only the required files for development will appear within the view. This will allow the subversion import to be quicker and easier to perform. After the CONFIG SPEC has been applied, the `svn import` command will be used. SVN import allows for local folders to be imported into an existing repository (a brand new, blank repository in this example). The exact commands used can be found below:

```
cd /views/GATE_view/GATE_vob/dev_vob
svn import . http://www.company-website.com/svn/new_repo/trunk -m"Initial Development Stream Migration"
```

The first command simply changes into the correct directory. The second command allows Subversion to pickup all the files within the directory (which has been filtered due to the customised CONFIG SPEC) and place them into the repository located at the specified address (this can be a local or remote repository). `svn import` will continue running until all files specified have been imported into the repository. After this had been completed, GATE developers were able to continue with their development work within Subversion.

The next step was to perform a full history migration. Before this occurred, GATE changed their CONFIG_SPEC back to the default ClearCase CONFIG_SPEC using the command below:

```
cleartool setcs -default
```

GATE created another folder to store their migrated history. This will mean that it is easy to inform developers exactly where their history will be within the new Subversion repository. The steps used to do this can be found below.

The senior GATE developer ran a checkout of the entire SVN repository with the following command:

```
svn checkout http://www.company-website.com/svn/new_repo
```

The senior developer created a new folder within the Working Copy called 'History'. This folder was then added and committed to the Subversion repository with the commands found below:

```
svn add History
svn commit History -m"Added new history storage folder"
```


The next and final stage was to use Migrate2SVN to migrate the complete set of folders, files and history from *dev_vob* to the new History folder within Subversion. After the `_migrate2svn.cfg` configuration file had been setup correctly and access to the VOB had been shut off, it was time to run Migrate2SVN.

These stages have been documented above. For more information on running Migrate2SVN, see the documentation for 'Simple Migration' in this User Guide.

Migrating Labels to Tags

Migrate2SVN will automatically migrate all labels and create Subversion tags. However, in some circumstances files that have been removed from the main line since the label was created may not be migrated correctly. In addition, if a labelled element exists on a branch that is not included in the migration, the element will not be migrated.

You are therefore strongly advised to provide Migrate2SVN with configuration specs for each label you would like to migrate. This will ensure complete accuracy between the label in ClearCase and the tag in Subversion.

 You must ensure that the config spec provided never eclipses the top-level view directory specified in your Migrate2SVN configuration file. If this happens, the config spec used for the main view may become corrupted.

Create a file for each configuration spec to be used, and create a configuration spec list file defining the purpose of each configuration spec.

Example `cspec_list_file`:

```
# <config_spec> @ <label> @ <svn_target_path> @ <commit_message>
config_spec_1.txt @ FIRSTLABEL @ @ Commit message 1
config_spec_2.txt @ SECONDLABEL @ @ Commit message 2
```

⚠ `<label>` and `<svn_target_path>` are mutually exclusive. One (but not both) of them must be provided for each line.

When migrating labels, you typically set `<label>` to the label to be migrated and provide a config spec that gives you a complete view of the label.

The `<svn_target_path>` allows you create a tag from any configuration spec, i.e. the contents of a view created by a particular configuration spec.

```
# <config_spec> @ <label> @ <svn_target_path> @ <commit_message>
config_spec_3.txt @ @ tags/product1/first_tag @ Commit message 1
```

If the config spec in `config_spec_3.txt` points to a config spec for the label `FIRSTLABEL`, the `cspec_list_file` above will rename the label to the tag "first_tag" in the subdirectory "product1".

⚠ `Migrate2SVN` will generate a temporary label based on the `<svn_target_path>` and copy all elements visible in the supplied config spec into the tag specified, so it is important to make sure the config spec provides only what you need.

Migrating Branches

`Migrate2SVN` migrates ClearCase branches into Subversion branches. ClearCase allows for hierarchical branches, but Subversion does not use this model for its branching. To cope with this, `Migrate2SVN` identifies branches and sub-branches within ClearCase and migrates the data found to the branches folder at the top of the Subversion repository. An example of how this works can be found below:

- A file called `bar.c` exists in the ClearCase VOB on `/main`
- It has been branched onto a branch called `/main/B1`
- It gets modified
- It has then been branched from `/main/B1` to `/main/B1/B2`
- Meanwhile `bar.c` has been modified on `/main`
- It has then been branched onto `/main/B2`

This leaves `Migrate2SVN` with the situation where there are two branches, both called `B2` but are considered separate branches by ClearCase. To translate this into Subversion, `Migrate2SVN` will name the branches as follows (this is how they will appear within the `Branches` folder of the new Subversion repository):

- branches
 - main
 - B1
 - B1_B2
 - B2
- tags
- trunk

As can be seen from the above, `Migrate2SVN` will label hierarchical branches within separate folders, all at the root of the Subversion repository.

Applying Subversion Properties During Migration

`Migrate2SVN` allows you to automatically apply Subversion properties to your files based on file extensions. you can create an `autoprops.txt` file and point `Migrate2SVN` to this file using the `svn_autoprops_file` configuration options.

```
svn_autoprops_file=/home/user/autoprops.txt
```

An example of an `autoprops` file is as follows:-

```

*.bat = svn:eol-style=CRLF;svn:executable
*.c = svn:eol-style=LF
*.cpp = svn:eol-style=native
*.css = svn:mime-type=text/css
*.doc = svn:mime-type=application/msword;svn:needs-lock=yes
*.dsp = svn:eol-style=CRLF
*.dsw = svn:eol-style=CRLF
*.erl = svn:needs-lock=yes
*.fmb = svn:needs-lock=yes
*.fmt = svn:keywords="Id Rev Date Author URL";svn:needs-lock=yes
*.gif = svn:mime-type=image/gif
*.gz = svn:mime-type=application/x-gzip

```

There is a full example of an autoprops file the config directory of the installation under "config/config.autoprops".

Using the Data After Migration

Once the migration has been completed, you will have a Subversion repository containing the history information from the original ClearCase repository. The repository will have the following structure:-

- branches
 - This contains all branches migrated, which typically includes the "main" branch.
 - Note that migrated branches in the Subversion repository only contain elements that were actually branched in ClearCase.
- tags
 - You will find a tag for each migrated label.
 - Automatic labels (created using the cspec_list_file option) are also place here.

Migrated Branches

Migrated branches only contain elements from ClearCase that were branched. Note that due to the differing concepts in ClearCase and Subversion, any intermediate directory levels required to place a branched file onto a branch will appear in the Subversion branch, even though the directories themselves were never branched. This cannot be avoided.

Migrated Tags

Tags in Subversion match the respective tags in ClearCase. You should be able to use these tags to build your software or as the basis for new development in the same way as you would have done in ClearCase.

Retrieving Log Information

ClearCase history has been preserved wherever possible in the Subversion repository. There are a small number of exceptions from this rule:-

- Directory checkin comments are not transferred. This is due to the fact that Subversion only versions files, not directories.
- Rename operations are not traceable through the Subversion repositories. All history entries will refer to the "final" name, not previous names the element may have had.
- Tags are created gradually as Migrate2SVN encounters the labels in the ClearCase history. They are not created in the order in which they were created in ClearCase.

To access history in Subversion, please use the "svn log" command. When you run "svn log" on the root of the repository, it will show you the history of the entire repository. When you run "svn log" on an individual file, it will only show the parts of the history that are relevant to the file in question. You can use the "-v" option for extra information on "svn log".

Working with Migrated Branches

In ClearCase elements are branched individually, and you typically combine a branch with another branch, the main line or a label using a suitable configuration spec. For example, the following configuration spec picks up elements from the branch, anything that has not been branched is picked up from the main line.

```
element * CHECKEDOUT
element * /main/branch1/LATEST
element * /main/LATEST
```

In Subversion, branches are fixed and span the whole repository (e.g. the whole trunk, a tag or another branch). However, since Migrate2SVN cannot guess the common branch point for the complete branch, it only places branched elements in the Subversion branch.

Consequently, migrated branches in Subversion will appear "incomplete". You will need to combine them with something else as you would in ClearCase with a configuration spec.

To create a new branch in Subversion, which is suitable for ongoing development, you need to branch the base versions (e.g. /main/LATEST) and then merge the branched elements on top. The following set of commands provide an example for a Subversion repository hosted at <http://my.company.com/svn/repo>:-

```
svn copy http://my.company.com/svn/repo/branches/main
file:http://my.company.com/svn/repo/branches/new_work_branch
svn co http://my.company.com/svn/repo/branches/new_work_branch
cd new_work_branch
svn merge --reintegrate http://my.company.com/svn/repo/branches/branch1
svn commit
```


The branch "new_work_branch" is now based on "main" plus "branch1" (applied on top), and it can now be used to build from or perform further development.

If your branching strategy in ClearCase was based on labels, then please change the above set of commands to branch off a tag rather than the main branch.

Migrating to Git

The Git distributed version control system already allows you to integrate with Subversion repositories. Users wishing to migrate from ClearCase to Git can therefore migrate to Subversion first and use Git's Subversion integration to migrate their data to Git.

Git's Subversion integration is able to extract the complete Subversion history into a Git repository.

 However, please note that due to the nature of a ClearCase to Subversion migration, Git's Subversion integration will not be able to convert branches and tags from a Subversion repository created with Migrate2SVN. Please see section "Why Are Tags and Branches Not Migrated to Git" for details.

The basic steps are:

- Create an empty Git repository
- Point your Git repository to a Subversion server
- Fetch the repository contents from Subversion
- Rebase your workspace

Subversion Repository Migration

This section describes in simple steps how to migrate a Subversion repository to Git.

You start by creating an empty Git repository.

```
mkdir repo
cd repo
git init
```

The next step is to point your Git repository to the Subversion server.

```
git svn init http://my.server.com/repo/trunk --stdlayout
```

Note that in this case the `--stdlayout` option was used to indicate that the repository uses the standard location for `tags`, `branches` and `trunk` directories. As a result, the contents of `trunk` will be migrated to the top of the Git repository.

However, please note that tags and branches will not be migrated to Git owing to a limitation in the previous migration process from ClearCase to Subversion. See section "Why Are Tags and Branches Not Migrated to Git" below.

The next step is to fetch the contents of the Subversion repository into your Git repository.

```
git svn fetch
```

At this point, depending on the size of the repository, you may want to get a cup of coffee, have dinner or embark on a Caribbean Cruise.

Once Git has fetched the complete repository contents, you can refetch any changes on a regular basis. However, please note that `git svn` will not update your workspace until you run the following command:-

```
git svn rebase
```

Why Are Tags and Branches Not Migrated to Git


Git's Subversion integration (`git-svn`) allows you to migrate tags and branches from a Subversion repository into Git. However, as the Git branching model is effectively represented by a pointer to a particular revision in the Git repository, `git-svn` needs to be able to trace back any branches or tags in the Subversion repository to a single revision on the Subversion trunk.

When migrating from a ClearCase repository, where tags and branches are made up of a combination of file revisions, tags and branches need to be built up incrementally, and whilst individual file revisions can be traced back to revisions on the trunk, there is no overall revision on the trunk that represents a tag or a branch in its entirety. As a result, `git-svn` will not be able to perform this mapping and tags and branches will not be visible within Git.

Please note that this limitation is due to the nature of a migration from a file-based configuration management system into a repository-wide-revision based system and cannot be overcome without introducing other, much more significant, limitations.

Migrating to Mercurial

The Mercurial distributed version control system already allows you to import data from Subversion repositories (including history). Users wishing to migrate from ClearCase to Mercurial can therefore migrate to Subversion first and use Mercurial's **convert** function to migrate their data into Mercurial.

 Note: This guide is relevant to a Migration where the default of creating tags and branches directories has been used in the `Migrate2SVN` config file

In order to successfully Migrate data from Subversion to Mercurial, a slight modification of the Subversion dump file (created by `Migrate2SVN`) is required to allow the process to succeed.

- `Migrate2SVN` does not include data/time and user information (in the Subversion dump file) on the first commit, and while this is not a problem for Subversion, it causes the `convert` function of Mercurial to fail.

The basic steps are:

- Modify the Subversion dump file
- Load the dump file into subversion (as normal)
- Run Mercurial `convert`

Subversion Migration

- Modify the Subversion dump file, replacing the following lines:

```
Revision-number: 1
Prop-content-length: 10
Content-length: 10

PROPS-END
```

- with:

```
Revision-number: 1
Prop-content-length: 129
Content-length: 129

K 7
svn:log
V 23
Migrate2SVN Directories
K 10
svn:author
V 11
Migrate2SVN
K 8
svn:date
V 27
1970-01-01T00:00:01.000000Z
PROPS-END
```

- Load the Subversion dump file as normal
- Run Mercurial convert
- Note: Mercurial convert has many options for converting; which are not discussed here

```
hg convert <<<Subversion_repo_name>>
```

The mercurial repository name will (by default) be the same name as your Subversion repository, with '-hg' appended to the repository name.