Summary of tasks and responsible parties 2

Migrating a Subversion project to another repository 3

Creating a Subversion dump file 4

Filtering a Subversion dump file 6

Importing a dump file into another Subversion repository 7

Updating development environments 8

Using with the command line 8

Updating your project with TortoiseSVN 9

Updating Eclipse environments with Subversive 11

Updating Eclipse environments using Subclipse 12

Updating Ant build files 15

Subversion repository migration workflow 16

Risk analysis 17

Summary of tasks and responsible parties

Task	Responsible Party
Coordinate with CBIIT development teams, general public, and all other interested parties regarding migration	CBIIT SVN Admin, local project manager
Coordinate user accounts (i.e. create new accounts in new repository for existing accounts in old repository)	CBIIT and ESN SVN admins
Ensure both machines have the necessary storage space to hold resultant dump files	CBIIT and ESN SVN admins
Create SVN dump file of CBIIT SVN project	CBIIT SVN Admin
Filter dump file so as to capture desired CBIIT project	CBIIT SVN Admin
Coordinate with CBIIT SVN Admin regarding space requirements for SVN dump file(s)	ESN SVN Admin
Load SVN dump file into ESN SVN repository	ESN SVN Admin
Notify developers	Project Manager
Update development environments and Ant build files	Developers
Update Anthill configuration	BDA team

Migrating a Subversion project to another repository

The process of migrating a project from one SVN repository to a different SVN repository requires a number of steps, namely:

- Creating a binary “dump” file containing the contents of an entire SVN repository
- Filtering out unneeded projects from the binary file
- Importing the contents of the binary “dump” file into another SVN repository

During the migration process a series of binary files are created, which happen to be quite large; consequently, ensure your working directories contain enough space to handle these files. This point cannot be over emphasized! Dump files contain *everything* in a repository; that is, all branches, tags, and meta data. Dump files *may* approach multiple terabytes in size. What’s more, these files appear to be human readable; however, it is imperative they not be opened via text editors as there is a good chance this will unfortunately (and unintentionally) corrupt the format of the file and thus not permit proper importing later.

The migration process also requires access to the machines both SVN repositories reside on. Unlike most SVN commands that work with URLs (and thus permit remote actions), the migration process uses a number of commands (that utilize file and directory paths) that must be run on the machine for which the intended repository resides.

Most importantly, the migration process must happen in coordination with development teams and the general interested community (that is, Quality Assurance personnel, other systems teams, management, and volunteers, for example). Once the migration starts, subsequent changes to the original SVN repository will not be reflected in the newer repository. Consequently, developers should be aware of a “blackout” period when they can no longer issue changes to a particular code base; what’s more, they should be given the new repository URL information as soon as it is available as they’ll have to do a new checkout and begin working on that code base going forward.

Before you start

☐ Do you have access to the machine where the project currently resides?
☐ Do you have access to the machine where the project is to reside?
☐ Coordinate with developers who use this repository-- they should not make anymore changes to the repository once you are ready to migrate.

Creating a Subversion dump file
Subversion permits the creation of dump files, which contain the entire contents (including historical information) of an SVN repository. These dump file then may be imported into another SVN repository. Thus, an entire project can be moved into another Subversion repository with all historical information and versions preserved.

It is important to note, however, that the process of creating a dump file works at a coarsely grained level; that is, a dump file can only be originally created for an entire SVN repository. Consequently, if your SVN repository has multiple projects, then the dump file will contain everything with regards to those projects (and the dump file will be quite large too). After a dump file is created, however, you can then filter out unneeded projects (i.e. projects you do not wish to migrate).

Before you can create a dump file, you must do two things. First, ensure you can successfully run the `svnadmin` command. Open up a command window on the machine that hosts the intended Subversion repository and type the following command:

```
$ svnadmin help
```

If the command works (you should see a list of available subcommands) then your user has the proper rights to create and filter dump files.

Next, as a dump file contains the contents of all projects residing in a Subversion repository (and it is assumed not all projects require migrating to another Subversion repository), it is imperative you capture the proper name of a desired project. Therefore, in the same command window, you'll need to execute the `list` command for a particular repository to obtain a listing of all projects within that repository. The list command is executed as follows:

```
$ svn list <svn url>
```

As you can see, the list command takes a Subversion repository URL (such as https://gforge.nci.nih.gov/svnroot). The output of this command is a list of projects-- find your project (such as caarray2) and ensure you copy the name down exactly.

Before you move on

- ☐ Do you have proper permissions to execute `svnadmin`?
- ☐ Do you have the name(s) of the project(s) you wish to migrate?

With these two steps complete, you are ready to create a dump file. Please ensure you have ample space in the directory in which you are working-- dump files are enormous.

As a side note, it is probably helpful at this point to determine how much size the current file system is utilizing -- this information can then be provided to the new repository administrators so as to better inform them of their impending space requirements.

To ascertain the amount of space a file system is using, simply type

```
$ du -sk
```

This command will provide a size in bytes. Keep this number handy as the new Subversion administrator might request it.

Creating a dump file requires one command. In the same command window, enter the following command:

```
$ svnadmin dump /directory/path/to/repo/ > repodumpfile
```

As you can see, the `dump` subcommand take a physical path to the repository. What's more, you must redirect the output into a file (via the `>` symbol). In this case, the contents of the repository are dumped into a file named `repodumpfile`.

The size of the dump file will depend on the size of the repository; suffice to say, small repositories can generate files approaching 1Gb.

Before you move on

- ☐ Was a dump file created?
- ☐ Did you write down the size of the current SVN repository file system?

Filtering a Subversion dump file
Subversion dump files contain the entire contents of a repository and in many cases, a migration process may only require a particular project from that repository; consequently, Subversion offers a filtering utility that will strip out unneeded information from a dump file.

To remove unneeded projects from a dump file, you'll need the name of the project (or projects) you wish to keep. With that proper name, in a command window, you can issue the following command:

```
$ svndumpfilter include project-name < repodumpfile  > projectname-dumpfile
```

As you can see, the `svndumpfilter` command reads a dump file and, in the process, creates a new dump file. You must provide a project name to the `include` subcommand-- this is the canonical name obtained from the `svn list` command earlier. The resultant dump file will be much smaller (relatively speaking, however) and only contain files and all related metadata for the intended project. If you wish to preserve multiple projects in a dumpfile, you can either run the svndumpfilter command using the include subcommand to include one or more projects separated by spaces or you may use the exclude sub command to exclude one or more projects like so:

```
$ svndumpfilter exclude project1 project2 < repodumpfile  > anotherdumpfile
```


Before you move on

☐ Was/were your filtered dump file(s) successfully created?

With your dump file(s) created, the next step is to compress them using `gzip`. This command is executed as follows:

```
$ gzip dumpfile
```

This process will result in a file named `dumpfile.gz`, which will be compressed and ready for moving on to another machine. Please note, as these files are large, emailing them will most likely not work.

At this point, you must coordinate with the administrator of the machine where the target repository resides-- you'll need to copy this zipped file onto the other machine. Coordination is key due to space requirements. Please note, this process might take an extended amount of time. Copying a multiple terabyte file requires a lot of bandwidth.

If this compressed dump file turns out to be large; that is, it is multiple Gigabytes, for example, it might make more sense to manually transfer the file via a storage mechanism, such as hard-drive.

Importing a dump file into another Subversion repository

Once your dump files are residing on another machine where your intended new SVN repository resides, you can import them via the `svnadmin load` command (if you used `gzip` to achieve the files, don't forge to use the `gunzip` utility to unzip them).

In order to load the contents of a dump file into an existing Subversion repository, you must, once again, have the proper privileges to successfully execute `svnadmin` commands on the machine where the intended repository resides. You must also know the physical directory path to the intended repository. Lastly, the dump file must also reside on the intended machine.

Before you start

- ☐ Did the `svnadmin` command successfully run?
- ☐ Do you know the physical path of the repository residing on the machine?
- ☐ Are your dump files on the machine?
- ☐ If the file(s) is/are compressed did you uncompress them (such as with `gunzip`)?

If these aspects are met, then you can issue the following command, which will import a top-level project (such as `caarray2`) including all its version files and history into another repository.

```
$ svnadmin load --force-uuid /directory/path/to/repo/ < your-dumpfile
```

The process takes as input your dump file. Once this is complete, you can verify things worked as expected by issuing a `svn co` command against the new repository URL like so:

```
$ svn co https://new.repo.url/project-name
```

If that worked, then you are done! Now, let dependent development teams know the new repository URL so they can begin using it.

Before you finish

- ☐ Are dependent development teams aware of the new repository URL?

Updating development environments

Migrating one Subversion code base to another Subversion repository does affect development teams. The migration process will force teams to stop checking code into one repository and enable them (after some period of time) to check code into a new repository.

Using with the command line

By far, the easiest method for reconfiguring a development environment is to utilize the command line; specifically, to invoke the `svn switch` command.

Before you start

☐ Do you know the new Subversion repository URL?

To utilize the `switch` command, you must know the old and new repository URLs. To execute the command, open up a command window in the root of your Subversion repository working copy. Next, type the following command:

```
$ svn switch --relocate <old repository URL> <new repository URL>
```

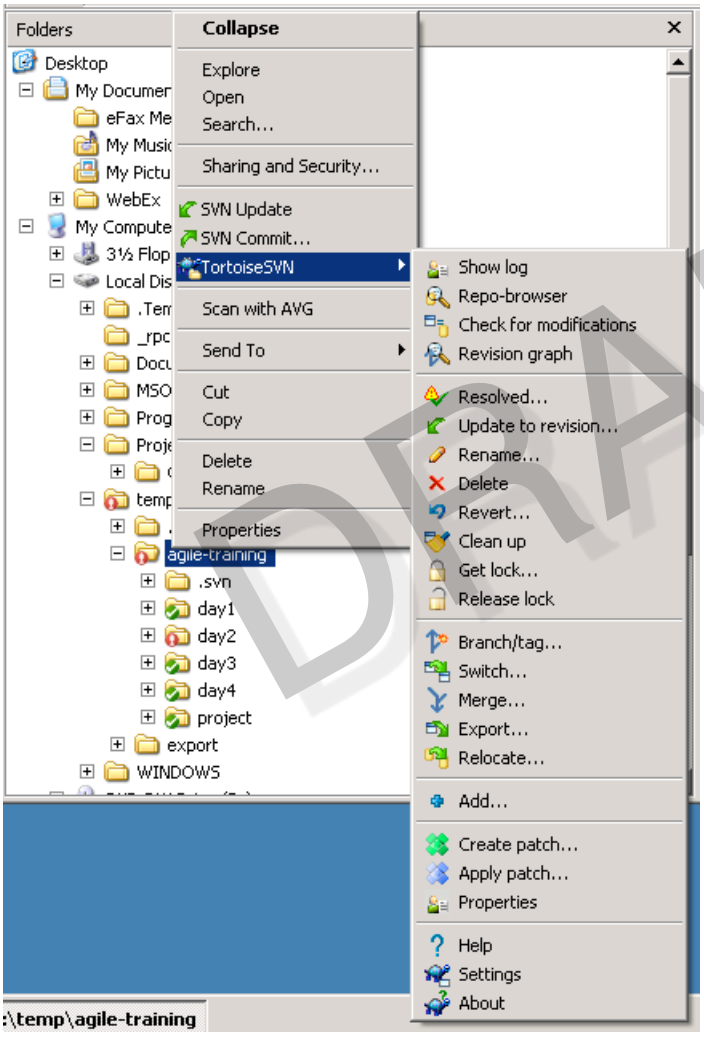
Updating your project with TortoiseSVN

If you utilize TortoiseSVN, you can easily update your project using the **Relocate** option.

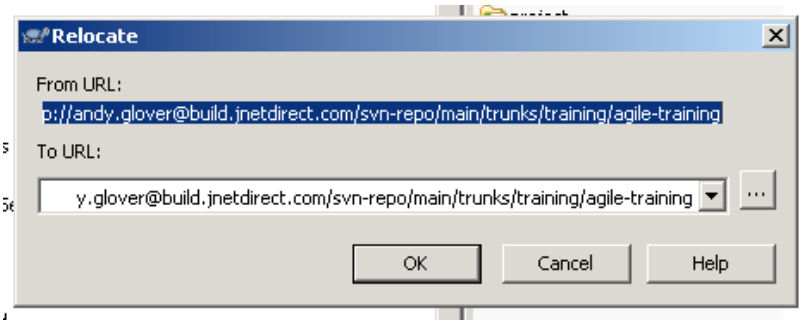
Before you start

☐ Do you know the new Subversion repository URL?

To relocate a Subversion repository, simply right click in the root directory of a working copy containing your project and select the **Relocate** option.



You'll be presented with a dialog box that will prompt you to provide the name of the new Subversion repository.



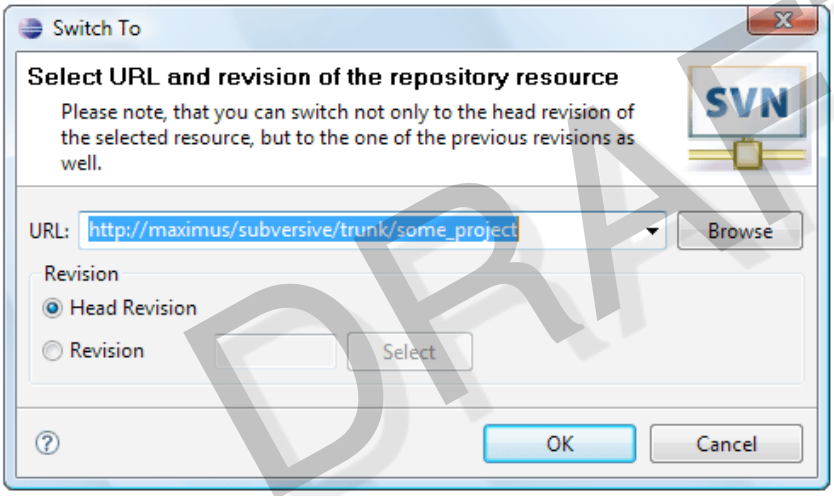
Updating Eclipse environments with Subversive

The Eclipse Subversive[1] plug-in facilitates working with a new repository via the **Switch** option.

Before you start

☐ Do you know the new Subversion repository URL?

To relocate a Subversion repository, simply right click in the root directory of a working copy containing your project and select the **Team**, then **Switch** option. A dialog window will appear, like the one shown below. Enter in the new repository URL.



[1] http://www.eclipse.org/subversive/

Updating Eclipse environments using Subclipse

Because Subversion uses the local file system uniquely with a specialized database, the level of effort to work with a new repository via Subclipse takes essentially one step; that is, once the code has been relocated, teams will need to execute a new checkout of the entire code base into their working sandbox area. Note, this means that the old sandbox area where developers coded locally goes away as that sandbox will remain configured to work with the previous version of the repository.

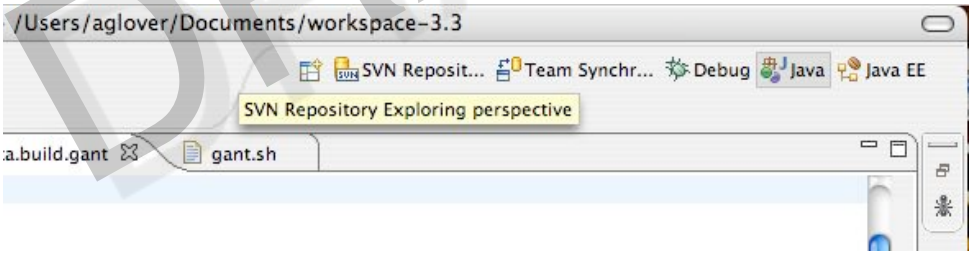
Before you start

☐ Do you know the new Subversion repository URL?

Assuming development teams were notified that a migration did take place and there is no remaining code that doesn't exist in the new repository because it wasn't checked into the old one before the cut off time, the process for working with a new repository location is as follows:

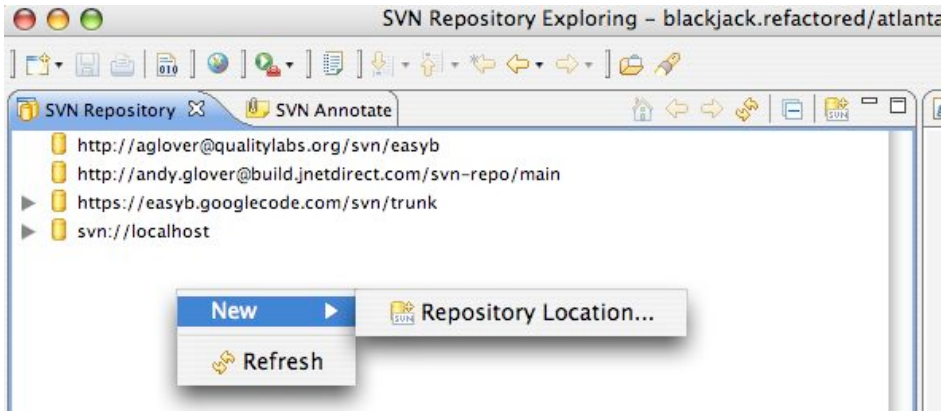
- Start over with a complete check out of the code base from the new repository
  - you can't update a project to point to a new SVN repository as it has local database configured with old one.

In Eclipse, this is done using the Subclipse plug-in[2]. If the plug-in has already been installed, check the **SVN Repository Exploring perspective.**

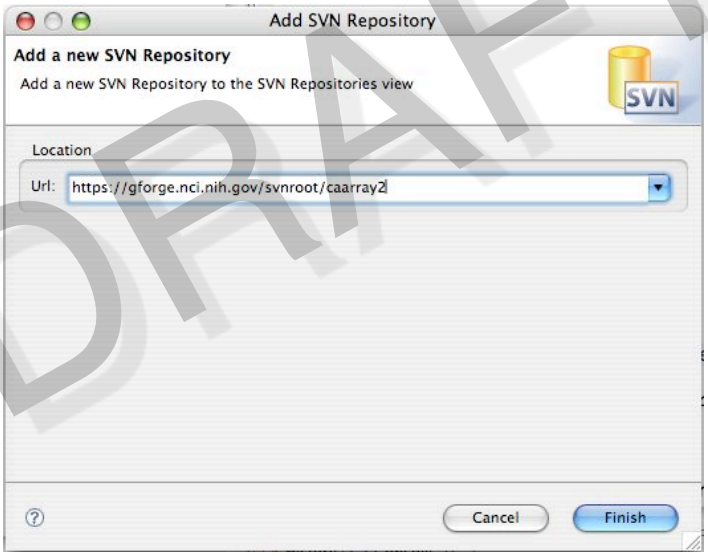


[2] http://subclipse.tigris.org/

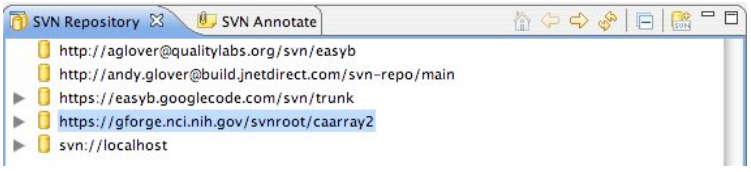
Next, you'll need to add a new repository; accordingly, right click SVN Repository explore pane and select **New Repository Location**.



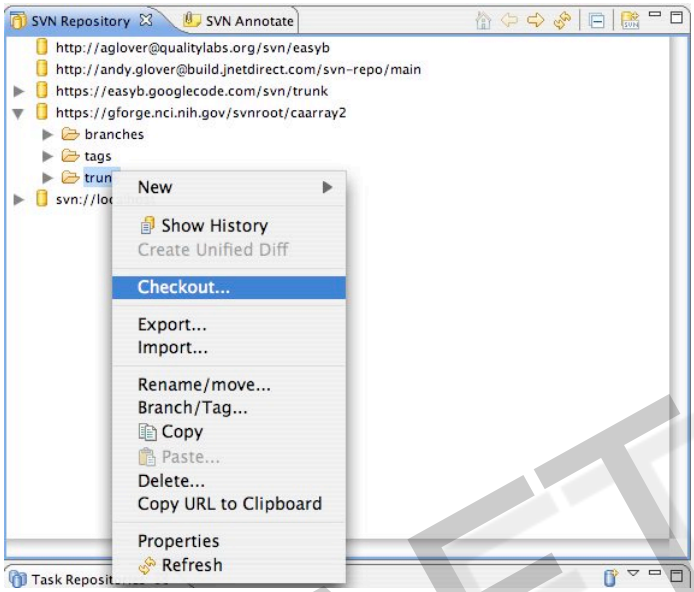
An **Add SVN Repository** dialog box should be presented-- go ahead and enter in the new URL for your Subversion repository.



Once the repository has been linked with your Eclipse environment, you will see it listed in the repository browser pane.



You can either right click the entire repository location or expand it to include sub directories and then right click to do execute a checkout.



It is important to note that you should not need to use the older repository going forward. In fact, it is best if you delete the old Eclipse project altogether to avoid accidentally working on it.

Updating Ant build files

The change in Subversion repository locations might affect build files; accordingly, it is imperative that each developer ensure that references to the old repository URL be updated to reflect the new URL.

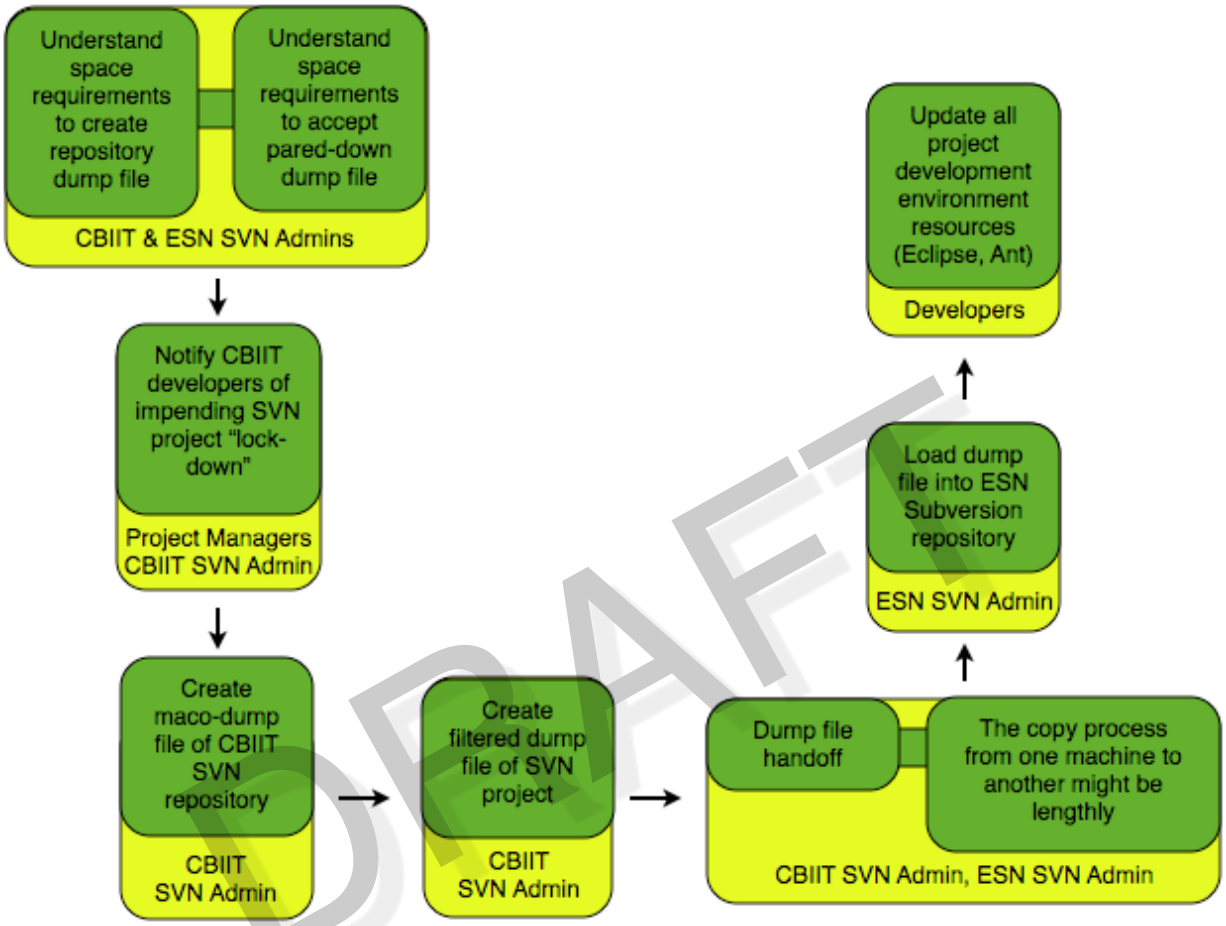
If you access to a command line utility, like GNU `find` and `egrep` you can quickly find out which files may contain references to the old repository URL like so (this is just one of a few different ways to obtain this information):

```
$ find . -iname *.* | xargs egrep -ci "<old URL>" | egrep -v ":0"
```

This command finds any file that contains the provided string (presumably the old repository URL) and produces the following output.

```
./project/build.xml:1
./project/foo/build.xml:2
./project/bar/build.properties:1
```

Subversion repository migration workflow



Risk analysis

It should be noted that while the process of migrating a project from one repository to another has been documented here and verified in controlled environments, there are associated risks. The risks are related to aspects of the migration and not necessarily the migration *itself*.

For instance, the following risks have been identified:

- Inadequate space for dump files
  - dump files for some Subversion repositories might exceed a terabyte in size. This is a large file; consequently, both machines (that is, the original machine hosting the repository and the other machine hosting the new repository) must have adequate space to handle such files.
- Inadequate bandwidth for dump file copies
  - As previously noted, Subversion dump files can become quite large. Moving a dump file over a network to another machine will consume substantial bandwidth. Depending on network speeds and the size of a dump file, the time to execute a complete move might require a tremendous amount of time.
  - Errors with respect to such a large network copy are possible; consequently, an adequate backup and recovery system should be ensured
- Downtime
  - The process of moving a dump file from one repository to another does require that those individuals working on the old repository *stop all activity with respect to Subversion* until such time as the new repository has been brought online. In a perfect world, this downtime will be limited; however, as already noted above, should an error occur or the copy process takes a lot of time, the risk of a prolonged blackout of all work becomes a serious reality. This creates a high degree of inconvenience for all dependent personnel.
- General inconvenience
  - As noted in the beginning of this document, for some projects there might be a large number of individuals that require notifying of the Subversion migration. If these people aren't properly notified, there will be high degree of inconvenience as they become shutout once the old repository is shutdown. The inquires from these individuals will require time to service properly.