Quick intro to Subversion

T. S. Norvell and D. K. Peters
Engr 6806. Memorial Unversity of Newfoundland

September 14, 2007

1 What are Subversion and TortoiseSVN

Subversion (svn) is a revision control system. It can track changes to the files in your project and inform you of conflicts created when two developers make changes to the same file. We are running Subversion on tera.mun.ca, where it maintains a "repository" containing all source code that you submit to it.

How does Subversion help? Consider the following scenarios.[1]

- Akbar has just written a class. Jeff needs this class before he can complete his coding. Akbar could email the code to Jeff and all the other members of the team. This results in a lot of email and a lot of duplicated work as all the team members try to keep their working copies in sync. With Subversion, Akbar submits his completed class to the repository and all the other team members can easily synchronize their copies.

- Akbar and Jeff change different parts of file Navigation.java. Akbar could email his copy to Jeff who looks at the differences and produces a file reflecting both changes. Then Jeff emails this merged copy to all other team members. This requires Jeff and Akbar to be aware that they are both working on the same file at the same time. With Subversion, Jeff and Akbar both update their working copies just before they submit their changes to the repository and Subversion will automatically create a merged copy.

- Akbar and Jeff change the same part of the same file. In this case when the second one updates prior to submiting his code he will be informed of the conflict and asked to resolve it prior to submitting the code.

- Binky reports a bug in the released version. Akbar and Jeff are midway toward the next release. While they can fix the bug, it will be months before they have a version that is stable enough to release. With Subversion they can recreate the source code of the last release version, make the fix there, and send a service pack to their customers. They can also merge the change to the release version into their current version.

[1] A very good discussion of the basics of working with Subversion available at http://svnbook.red-bean.com/en/1.4/svn.tour.html.

- Akbar's hard-disk crashes. With Subversion only changes made since the last time he submitted changes to the repository are lost.

TorroiseSVN is a graphical user interface integrated with Windows Explorer that makes the Subversion client easier to use on Windows platforms. There are similar applications available on other operating systems (e.g., KDEsvn on Linux, and svnX on Mac). There are also plugins that integrate subversion with most of the major Integrated Development Environments (IDE).

2 Repositories and Working Copies

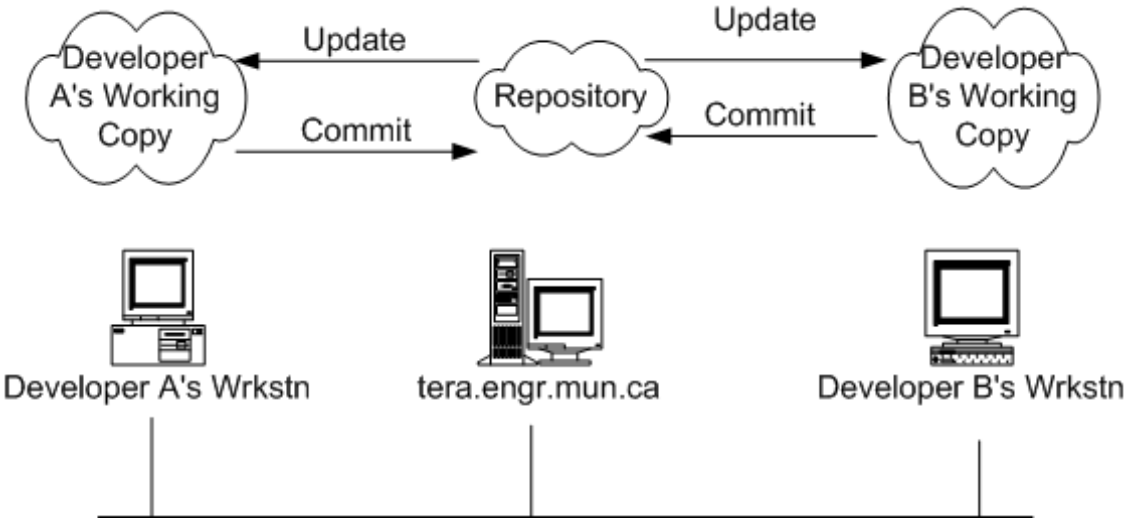

Figure 1: Repositories and Working Copies

When you use Subversion, the server keeps a copy, called a *repository* of all of your files and the history of every change that has been made to them. From this repository the server can produce any version of your files that has ever been checked in. Normally you want to use the latest version, which is called the head.

To work with your files on a particular computer you must first set up a local copy, known as the *working copy*, of some version (usually the most recent) of your repository. The working copy is just a normal directory and you can have as many of them as you need, including multiple copies on the same computer (just not in overlapping directory hierarchies). You can also have working copies of sub-trees of a repository in different locations (e.g., you could keep your java source working copy in a sub-directory of your eclipse workspace, and design documents elsewhere).

Any changes made to the working copy aren't reflected in the repository until you `commit` them.

3 Common procedures

3.1 Installing TortoiseSVN

Skip this step if TortoiseSVN is already installed.

- Obtain the appropriate TortoiseSVN installer from http://tortoisesvn.tigris.org/.
- Double click and follow the instructions. Reboot.

3.2 Getting a Working Copy

Before you can work with files in the repository, you must check out (`svn checkout`) a working copy. In TortoiseSVN this is done as follows:

- Create a directory in which you will store the working copy.
- Navigate to the directory.
- Right-click and select "SVN Checkout."
- For the repository enter `svn://tera.engr.mun.ca/6806/Team_A/` (replacing the last part with your appropriate team name), click "Ok".
- Enter your username and password.

3.3 Changes to the working copy

You can make changes to the files and directories now. Use whatever editor or other tools are appropriate, depending on the files being edited.

After you add any files or directories you must tell svn that they should be added to the repository and managed (`svn add`). In Tortoise SVN, right-click on the file and select `TortoiseSVN→add`.

To delete a file or directory you must tell svn to delete it (`svn delete`. The file is not actually removed from the repository, but it is marked so that it won't be included in later updates, and will be removed from other working copies upon update. Use `TortoiseSVN→delete`.

3.4 Updating

From time to time you will want any changes that others have made to the repository reflected in your working copy. To do this you must *update* your working copy (`svn update`). In TortoiseSVN, right-click on the root directory of your working copy (or any particular file or directory to do a partial update) and select `TortoiseSVN→update`.

In particular it is good practice to do an update before committing a block of changes so that any changes made by other users are merged into your changes. This is not strictly necessary since the commit will fail if any files that you have changed were changed by

someone else, but it can prevent problems. In particular, if two users are modifying code components where one depends on the other, it is possible for them to make incompatible changes and for those to be overlooked

3.4.1 Conflicts

A *conflict* in svn terminology happens when two users modify the *same part* of the same file. (Note: If they modify *different* parts of the same file then svn will merge the changes together when the second user updates her working copy. This is not a conflict.)

If you have made changes that cause a conflict, the update command will tell you. Watch for red lines like:
`Conflicted someFileName`
in the log window. Editing the file will show you where your changes conflict with others, with lines that look like this:

```
<<<<<<< .mine
Here is a 2nd line inserted in the middle.
=======
Here is the second line inserted in the middle.
>>>>>>> .r126
```

This says that your working copy contains the lines between `<<<<<<< .mine` and `=======` and the repository (in revision 126) has the lines between `=======` and `>>>>>>> .r126`. Your working copy will also contain three new files, as follows:

Extension	Description
`.mine`	Your working copy
`.r125`	The revision that you started from (rev. 125 in this case)
`.r126`	The current committed revision (rev. 126 in this case)

You must make your working copy right, and then tell svn that you've resolved the conflict (`svn resolve` or `TortoiseSVN→Resolved ...`). You can then commit the files (note that since a commit is atomic, you must commit everything again).

Not all mutually incompatible changes are considered by svn to be conflicts. For example suppose that a subroutine has been declared with declaration

```
int intpow(int exponent, int base)
```

Developer A and B start the day by updating their working copies. Both start with the same source. Developer A changes the order of the parameters in the declaration and definition. Developer B writes several calls to the subroutine using the original order. A updates (there is no change) and commits (see next section). Now B updates (getting A's changes) and commits. Svn will not notice any problem. However svn's definition of a conflict is good enough for most purposes.

3.5 Committing

From time to time you will want to share changes that you have made to your fellow developers. It is a good idea to do this whenever you have made significant progress that they

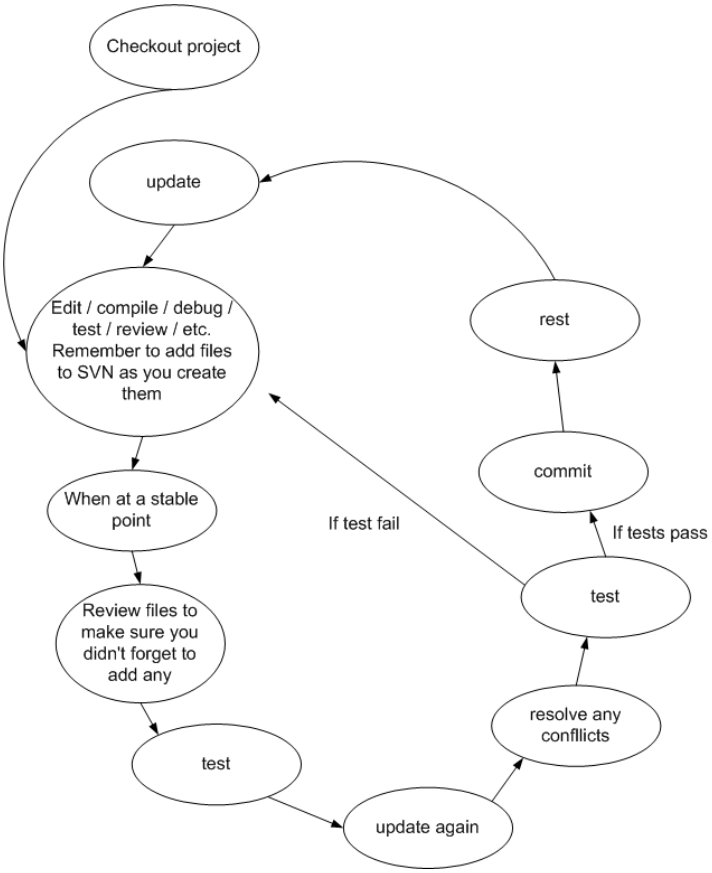

Figure 2: Typical svn usage

may benefit from. It is a bad idea to do this when your code does not compile or is otherwise in rough shape.

- First update your own working copy. Resolve any conflicts.
- Right click the root of your working copy.
- Select `SVN Commit ...`.
- Enter a log message describing the nature of the work you have done since your last commit.

4 Typical use of Subversion

Figure 2 shows typical usage of subversion. It is a good idea to break the work you have to do into small tasks each of which take the system from a stable point to a stable point. For example, adding one feature might be a task. Fixing one bug might be a task. Each task takes you once around the cycle.

The first update ensures that you are working on the most recent version of the system, that is that you have changes made by your teammates. After updating you complete the task.

The second update, right before the commit, is a good idea for several reasons. First it alerts you to any conflicts before you commit. Second, it will list all the files that you've modified, which reminds you of what you've done. Third, it lets you obtain any changes made by your colleagues while you've been working on the task, so you can then test your work with the most up-to-date versions of all files. I often run all unit tests right before and after doing the second update.

It is a bad idea to commit when the system will not compile. This tends to annoy your colleagues.

A common mistake with revision control is to forget to add a file that you've created. This typically results in a system that compiles for you, but no one else. Before you commit, make sure all new files have been added.

5 But Wait There's more

The above only scratches the surface of what you can do with Subversion and TortoiseSVN. You can compare your working version of a file with the repository version (diff); you can obtain the history of changes to a file (log); you can inform others of your intention to edit; you can have svn email you when someone has done a commit; you can recreate earlier versions of the module; you can create branches that represent parallel versions of the module; and so on.

For more information see the documentation at http://subversion.tigris.org/.

A free book on subversion is available at http://svnbook.red-bean.com/en/1.4/index.html.

No revision control system will replace communication between developers and careful project organization. Make sure that you all know what the other developers are doing (or at least which parts of the source they are working on).