

Using SVN to Manage Source RTL

CS250 Tutorial 1 (Version 092509a)

September 25, 2009

Yunsup Lee

In this tutorial you will gain experience using the Subversion (SVN) to manage your source RTL and code. You will be using SVN to submit your lab assignments, but more importantly SVN will make it much easier for groups of students to work on the same project simultaneously. SVN keeps track of the changes each user makes to the source RTL - this complete version history enables users to monitor what changes other users are making (each change includes log information), tag working versions of the design (so that users can always revert back to a working version), and resolve conflicts (when two users change the same bit of RTL at the same time).

SVN stores all version information in a central *repository*. Users cannot access the repository directly, but must instead use a special set of SVN commands. Users begin by *adding* the initial version of their source RTL to the repository. Users can then *checkout* the most current version of the source RTL into a private *working directory*. Local changes to the working directory are not stored in the repository until the user does a *commit*. Each commit includes a log message so that other users can easily read what has changed. If multiple users are changing the source RTL at the same time, then the state of the repository might be different from when the user performed the original checkout. At any time, a user can do an *update* which brings the checked out working directory in sync with the global repository. Sometimes an update will identify a *conflict*. A conflict indicates that two users have made changes to the same bit of of RTL. Users must resolve conflicts by hand - essentially they must choose whose changes should be permanent. Figure 1 illustrates the relationship between the central repository and each user's personal working directory.

In this tutorial we will create a test directory and a test file, add them to the repository, make some changes, commit these changes, and then emulate issues which arise when multiple users change the same file at the same time.

You can find more information in the SVN book (svn-book.pdf) located at the following website: <http://svnbook.red-bean.com>.

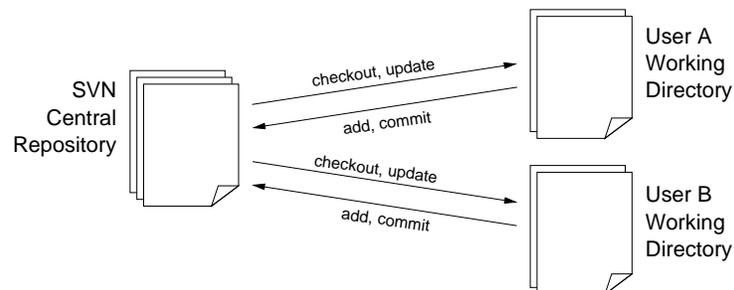


Figure 1: Basic SVN model. Users checkout versions from the central repository into private working directories.

Getting started

Before using the CS250 toolflow you must run the course setup script with the following command. The course setup script will set the `SVNREPO` to `https://isvn.eecs.berkeley.edu/cs250`. The SVN commands use this environment variable to determine the location of the repository.

```
% source ~cs250/tools/cs250.bashrc
```

Adding Directories and Files to SVN

In this section we will look at how to add directories and files to the repository. Figure 2 shows the timeline of SVN commands involved in this section.

All of your work will be in either your student SVN directory or your group's project SVN directory. For this tutorial we will be working in your student SVN directory. The very first step is to checkout this directory. All SVN commands are of the form `svn <command>`. For example, assuming your username is `yunsup` the following commands will checkout your student SVN directory.

```
% mkdir tut1
% cd tut1
% svn checkout $SVNREPO/yunsup
Error validating server certificate for 'https://isvn.eecs.berkeley.edu:443':
- The certificate is not issued by a trusted authority. Use the
  fingerprint to validate the certificate manually!
Certificate information:
- Hostname: isvn.EECS.Berkeley.EDU
- Valid: from Thu, 15 Jan 2009 19:53:47 GMT until Sat, 15 Jan 2011 19:53:47 GMT
- Issuer: Instructional Support Group, University of California Berkeley, ...
- Fingerprint: 41:10:d6:7a:ad:4b:a5:20:02:09:f6:65:ad:4c:f5:37:f3:ef:03:30
(R)eject, accept (t)emporarily or accept (p)ermanently? p
% cd yunsup
```

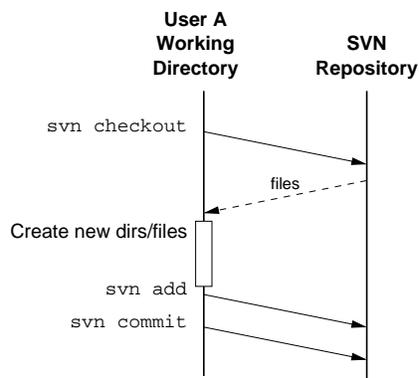


Figure 2: Timeline for checkout, add, and commit.

Whenever you access the SVN repository for the firsttime, it will complain about the certificate. This is because the certificate on the server is not issued by a trusted authority. Go ahead and accept the certificate permanently.

The following commands first create a test directory and a typical SVN directory structure, test file, and then add them to the SVN repository. The trunk directory is supposed to be the main development line for the project. The branches directory is supposed to be a place for branches. The tags directory is used for creating tags.

```
% mkdir svntest
% cd svntest
% mkdir trunk branches tags
% cd trunk
% echo "Fred : 510-555-0123" > phone-list.txt
% cd ../../
% svn add svntest
```

Although the directory and file are *added* to the repository they are not actually *in* the repository until you commit them. Adding directories and files simply lets SVN know that you want the versioning system to track these files. You need to *commit* the directory and file before they are permanently in the repository. You can use the following command to check the current status of all files.

```
% pwd
tut1/yunsup/svntest
% svn status
```

The status information should reflect that `phone-list.txt` has been locally added. Now you will commit your new files to the repository.

```
% pwd
tut1/yunsup/svntest
% cd ..
% svn commit svntest
```

The `svn commit` command takes a list of files and directories as an argument. The `svn commit` command is recursive, so committing a directory will effectively commit all files (and subdirectories) within the directory. If you do not specify any files or directories, then SVN will commit the current directory.

After executing the `svn commit` command, you will be able to enter a log message using your default text editor. You can change the default text editor by setting the environment variable `SVN_EDITOR`. You can add the environment variable to your default shell script so that the variable is set automatically in the future whenever you login.

```
% export SVN_EDITOR=vim
```

Use the `svn status` command again to verify that the `phone-list.txt` file has now been committed into the repository. The status information lists a revision number. Every change is given a

unique revision number. These numbers are assigned by the SVN system, and users should usually avoid working with revision numbers directly.

Our final step is to delete your working directory. It is essential that you always keep in mind the difference between what is in the repository versus what is in the repository. As long as all your new files have been added, and all your changes have been committed then there is nothing wrong with deleting your working directory and doing a clean checkout.

```
% pwd
tut1/yunsup
% cd ..
% rm -rf yunsup
```

Making and Committing Changes

In this section you will checkout the `svntest` directory, make a change, and then commit the change back into the repository. Figure 3 shows the timeline of SVN commands involved in this section. Our first step is to do a clean checkout of the `svntest` directory. The `svn checkout` command is recursive, so by checking out the `svntest` directory you also checkout all files and subdirectories contained within `svntest`.

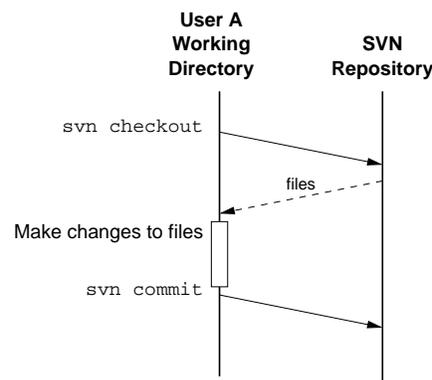


Figure 3: Timeline for checkout, modify, and commit.

```
% pwd
tut1
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
```

Now say Fred's phone number has changed.

```
% cd svntest
% perl -i -pe 's/Fred.*/Fred : 510-555-4567/' phone-list.txt
```

You can use the `svn diff` command to see how your current working directory differs from the repository. For example, if you use the `svn diff` command in the `svntest` directory, it will show that you have updated Fred's phone number. The revision number `X` is the number stored on the server.

```

% pwd
tut1/svntest
% svn diff
Index: phone-list.txt
=====
--- phone-list.txt (revision X)
+++ phone-list.txt (working copy)
@@ -1 +1 @@
-Fred : 510-555-0123
+Fred : 510-555-4567

```

And finally you commit your changes with an appropriate log message.

```

% svn commit

```

You are done making your changes and everything has been committed so you can now delete your working directory.

```

% pwd
tut1/svntest
% cd ..
% rm -rf svntest

```

Updating the Working Directory

In this section you will see how SVN can help multiple users work on the same project at the same time. Figure 4 shows the timeline of SVN commands involved in this section. To emulate two users you will checkout your `svntest` directory into two different working directories. You begin with User A using the following commands to checkout the current `svntest` directory.

```

% pwd
tut1
% mkdir userA
% cd userA
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd ..

```

Say User B now checks out the current `svntest` directory, adds a new file which contains email addresses, and adds information about a new person named Jane.

```

% pwd
tut1
% mkdir userB
% cd userB
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd svntest
% echo "Fred : fred@eecs.berkeley.edu" > email-list.txt
% echo "Jane : 510-555-0021" >> phone-list.txt

```

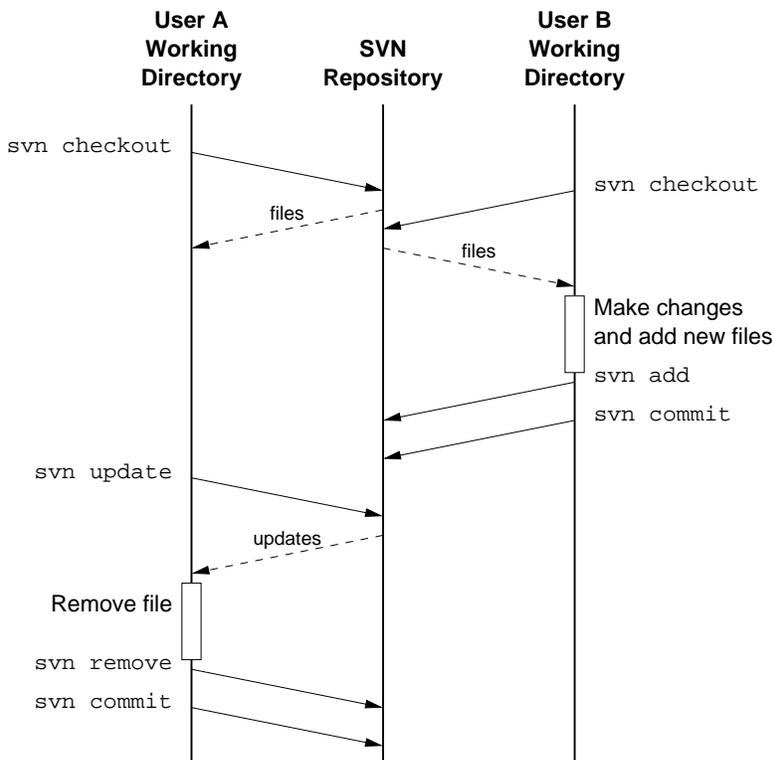


Figure 4: Timeline for two users with no conflicts

```

% echo "Jane : jane@eecs.berkeley.edu" >> email-list.txt
% cat phone-list.txt
Fred : 510-555-4567
Jane : 510-555-0021
% cat email-list.txt
Fred : fred@eecs.berkeley.edu
Jane : jane@eecs.berkeley.edu
% svn add email-list.txt
% svn commit
  
```

User B's final `svn commit` command commits both his changes to the `phone-list.txt` file as well as his newly added `phone-list.txt` file. Notice that at this point in time, User A's working directory is out-of-date (it does not contain the new `email-list.txt` file nor does it contain the updated `phone-list.txt` file). User A can use the `svn status --show-updates` command to realize that her working directory is out-of-date with respect to the central repository. If the User A doesn't use the `--show-updates` option then it prints locally modified items without accessing the central repository.

```

% pwd
tut1/userB/svntest
% cd ../../userA/svntest
% svn status --show-updates
*      X  phone-list.txt
  
```

```

*          email-list.txt
*      X   .
Status against revision:  Y

```

Notice that the `phone-list.txt` file and the `email-list.txt` file have an asterisk mark which means a newer version exists on the server. X is the local revision number, while Y is the revision number stored on the server. User A can use the `svn update` command to bring her working directory in sync with the central repository.

```

% pwd
tut1/userA/svntest
% ls -a
.svn phone-list.txt
% cat phone-list.txt
Fred : 510-555-4567
% svn update
% ls -a
.svn email-list.txt phone-list.txt
% cat phone-list.txt
Fred : 510-555-4567
Jane : 510-555-0021
% cat email-list.txt
Fred : fred@eecs.berkeley.edu
Jane : jane@eecs.berkeley.edu

```

If User A uses `svn status --show-updates` again, she will see that her files are now up-to-date with respect to the central repository. As another example, User A can use the following commands to delete a file and then restore that file to the most current version in the repository.

```

% pwd
tut1/userA/svntest
% rm -f phone-list.txt
% svn update

```

If User A really wants to delete the file from the repository, then she can first delete the file from the working directory and then use the `svn remove` command. These removals are not visible to other users until User A does a `svn commit`.

```

% pwd
tut1/userA/svntest
% rm -f email-list.txt
% svn remove email-list.txt
% svn commit

```

If User A wants to rename or move files, then she can use the `svn move` command.

```

% pwd
tut1/userA/svntest
% svn move phone-list.txt telephone-list.txt
% svn commit

```

If User B now does a `svn update`, his copy of `email-list.txt` will be removed and the file `phone-list.txt` will be renamed as `telephone-list.txt`. Finish up by deleting the working directories.

```
% pwd
tut1/userA/svntest
% cd ../../
% rm -rf userA
% rm -rf userB
```

Resolving Conflicts

In the previous section you examined what happens when two users simultaneously modify files in the same project. Notice however, that in the previous example, the two users never modified the same line of a file at the same time. In this section you will examine how to handle these *conflicts*. Figure 5 shows the timeline of SVN commands involved in this section.

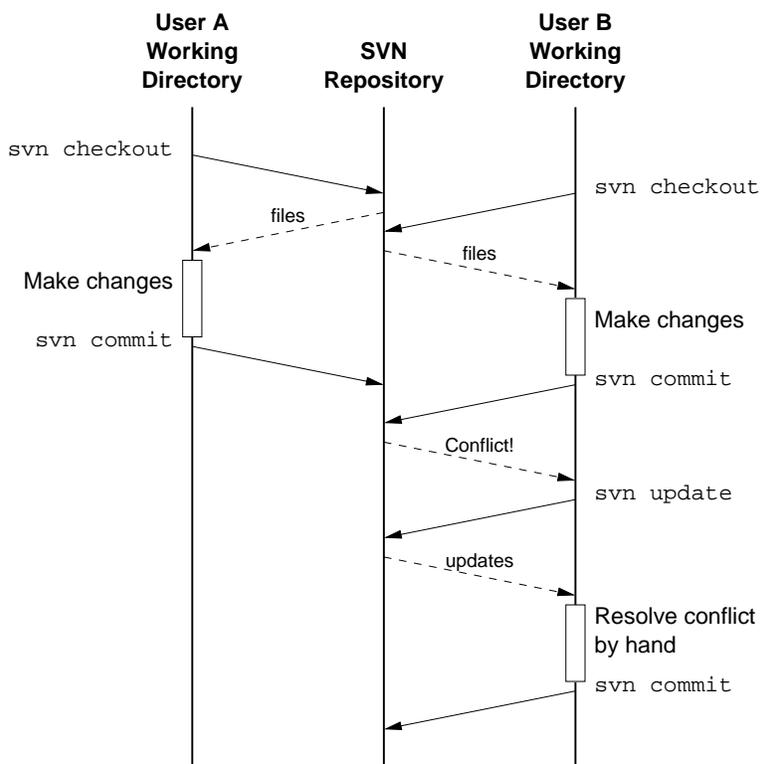


Figure 5: Timeline for two users with conflicts

Begin with both User A and User B doing a standard checkout.

```
% pwd
tut1
% mkdir userA
```

```
% cd userA
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd ..
% mkdir userB
% cd userB
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd ..
```

Now assume that User A changes Fred's phone number as follows.

```
% pwd
tut1
% cd userA/svntest
% perl -i -pe 's/Fred.*/Fred : 510-555-3333/' telephone-list.txt
% svn commit
```

Now assume that User B changes Fred's phone number to a different number.

```
% pwd
tut1/userA/svntest
% cd ../../userB/svntest
% perl -i -pe 's/Fred.*/Fred : 510-555-5555/' telephone-list.txt
% svn commit
Sending          telephone-list.txt
Transmitting file data .svn: Commit failed (details follow):
svn: Out of date: 'yunsup/svntest/trunk/telephone-list.txt' in transaction 'X-1'
svn: Your commit message was left in a temporary file:
svn:      'tut1/userB/svntest/svn-commit.tmp'
```

Notice that User B has received an error when trying to commit his changes. The SVN repository has realized that User B's change conflicts with User A's original change. So User B must do a `svn update` first.

```
% pwd
tut1/userB/svntest
% svn update
Conflict discovered in 'telephone-list.txt'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: e
```

The SVN messages during the update indicate that there is a conflict in `telephone-list.txt`. A conflict simply means that there is a file in the current working directory which has lines which differ from what is in the repository. SVN tries to merge files so that only true conflicts are reported. In other words, if two people change different parts of the same file, then usually SVN will be able to merge the two versions without a conflict. In this case both User A and User B changed the exact same line, so SVN does not know which modification should take priority. SVN requires that users resolve conflicts by hand. You can pick a way to resolve a conflict by selecting an option when SVN asks. Assume you picked the edit option. The text editor will show the following.

```

1 <<<<<<< .mine
2 Fred : 510-555-5555
3 =====
4 Fred : 510-555-3333
5 >>>>>>> .rX
6 Jane : 510-555-0021

```

The content involved in the conflict is delimited by <<<<<<< and >>>>>>>. Assume that User B wants to override User A's earlier change. Leave the text from the mine part, and delete everything else to make the following.

```

1 Fred : 510-555-5555
2 Jane : 510-555-0021

```

After saving the file SVN will prompt you again, but with a new option called resolved.

```

Select: (p) postpone, (df) diff-full, (e) edit, (r) resolved,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options: r

```

Now User B's commit has succeeded and essentially User A's changes will have been overridden. User A can then do a `svn update` to synchronize her working directory. A common rule of thumb is to always do an update (to catch conflicts) before doing a commit. Finish up by deleting the working directories.

```

% pwd
tut1/userB/svntest
% cd ../..
% rm -rf userA
% rm -rf userB

```

Using Tags

In the previous sections you have learned how to use SVN to manage various versions of your source RTL. You have primarily focused on how to manipulate the most current version in the repository. Sometimes when you reach a milestone, you want to mark a version so that you can retrieve it at a later time even if you have already made (and committed) many other changes. You can do this with SVN tags. A tag is simply a symbolic name you give to a specific version of several files.

The following commands checkout the `svntest` directory, add two new files, and then tags everything. It is important to always do a `svn update` and a `svn commit` before doing a `svn copy` since the tag operation works on the most current version in the repository. If you have made changes in your local directory which are not yet committed then this could incorrectly tag the files.

```

% pwd
tut1
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd svntest

```

```

% echo "Fred : fred@eecs.berkeley.edu" > email-list.txt
% echo "Jane : jane@eecs.berkeley.edu" >> email-list.txt
% echo "Fred : SODA 802" > office-list.txt
% echo "Jane : SODA 903" >> office-list.txt
% svn add email-list.txt
% svn add office-list.txt
% svn update
% svn commit
% svn copy $SVNREPO/yunsup/svntest/trunk \
           $SVNREPO/yunsup/svntest/tags/checkpoint

```

The above commands tag all three files (`telephone-list.txt`, `email-list.txt`, and `office-list.txt`) with the symbolic tag `checkpoint`. You can now retrieve this version of the files at any time by using this tag. To see how this works first commit some additional changes.

```

% pwd
tut1/svntest
% echo "Sara : 510-555-0234" >> telephone-list.txt
% echo "Sara : sara@eecs.berkeley.edu" >> email-list.txt
% echo "Sara : SODA 810" >> office-list.txt
% svn update
% svn commit

```

Now delete the working directory and try checking out two different versions of the files.

```

% pwd
tut1/svntest
% cd ..
% rm -rf svntest
% mkdir tagged-version
% cd tagged-version
% svn checkout $SVNREPO/yunsup/svntest/tags/checkpoint svntest
% cd ..
% mkdir current-version
% cd current-version
% svn checkout $SVNREPO/yunsup/svntest/trunk svntest
% cd ..
% cat tagged-version/svntest/email-list.txt
Fred : fred@eecs.berkeley.edu
Jane : jane@eecs.berkeley.edu
% cat current-version/svntest/email-list.txt
Fred : fred@eecs.berkeley.edu
Jane : jane@eecs.berkeley.edu
Sara : sara@eecs.berkeley.edu

```

The checked out version in the `tagged-version` directory corresponds to the `checkpoint` symbolic tag, while the version in the `current-version` directory corresponds to the most recent version in SVN. Tagging is particularly useful when you reach a working milestone. You can tag your project and then at any time you can go back and retrieve the working version as of that milestone. Finish up by deleting the working directories.

```
% pwd
tut1
% cd ..
% rm -rf tut1
```

Browse the Repository using your Web Browser

Type in `https://isvn.eecs.berkeley.edu/cs250/<username>` into your browser. Then use your credentials to login. Now you can browse your repository online. You will have the same certificate problem, though just allow the browser to accept the certificate permanently.

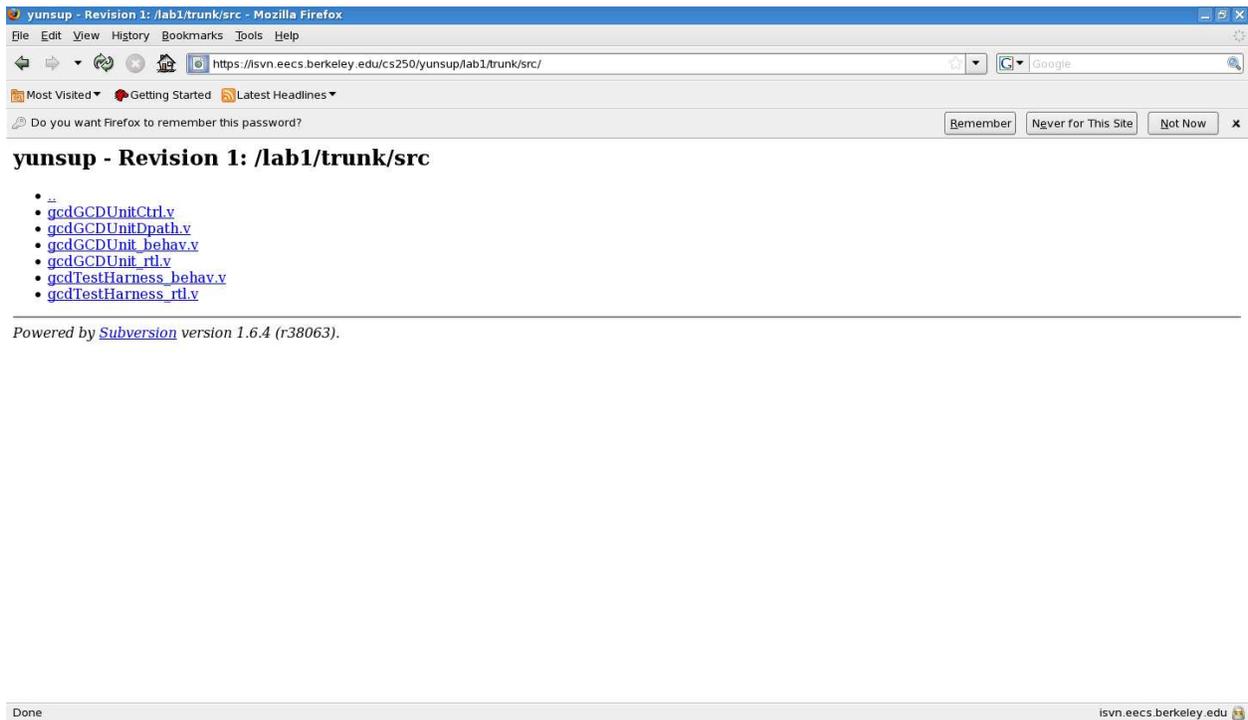


Figure 6: Browsing the Repository using your Web Browser

Review

In this tutorial you have learned how to use SVN to manage your source RTL. You have learned how to add new files to the repository, checkout files from the repository, and commit local changes into the repository. The following table lists the SVN commands that you will use most frequently in this course.

<code>svn checkout (co)</code>	Checks out files from the repository into the working directory
<code>svn add</code>	Adds new files to repository (must commit after adding)
<code>svn commit (ci)</code>	Commits files in working directory into the repository
<code>svn update (up)</code>	Brings working directory in sync with respect to repository
<code>svn remove (rm)</code>	Removes file from SVN
<code>svn move (mv)</code>	Rename files from SVN
<code>svn status (st)</code>	Shows status of files in working dir compared to current version in repo
<code>svn diff</code>	Shows how files in working dir differ from current version in repo
<code>svn copy (cp)</code>	Adds a symbolic tag for current version

Acknowledgements

Many people have contributed to versions of this tutorial over the years. The tutorial was originally developed for 6.375 Complex Digital Systems course at Massachusetts Institute of Technology by Christopher Batten. Contributors include: Krste Asanović, John Lazzaro, Yunsup Lee, and John Wawrzynek. Versions of this tutorial have been used in the following courses:

- 6.375 Complex Digital Systems (2005-2009) - Massachusetts Institute of Technology
- CS250 VLSI Systems Design (2009) - University of California at Berkeley