What is SVN

- SubVersioN is a centralised system for sharing information.
- It works on a client-Repository model
- Can be used for anything
- I use it for papers, for writing documents and for code
- It gives you versioned history and allows multiple simultaneous development.

Basic Work Cycle

1. import : `svn import /tmp/myproject https://www2.msm.ctw.utwente.nl/svn/repos/THORNTON -m "initial import"`
2. Checkout : `svn checkout https://www2.msm.ctw.utwente.nl/svn/repos/THORNTON my_project`
3. Add files `svn add foo3.c`
4. Update : `svn up` or `svn update`
5. Commit : `svn commit -m "added some changes"` or `svn co -F svn_log_file`

Updating and conflicts

```
$ svn update
A foo.cpp
U INSTALL
D foo2.cpp
G README
C bar.c
Updated to revision 46.
```

Updating and conflicts

```
$ svn update
...
C bar.c
Updated to revision 46.
```

You will will now have the following additional files

```
bar.c
bar.mine
bar.r43
bar.r46
```

Updating and conflicts

```
$ cat bar.c

int N;
N << cin;
for (int i=0;i++;i<10)
{
<<<<<<< .mine
cout << "The value entered was " <<N;
=======
cout << "You entered" <<N << "was the value you" << endl;
>>>>>>> .r46
}
cin << N
```

Updating and conflicts

```
$ cat bar.r43

int N;
N << cin;
for (int i=0;i++;i<10)
{
cout << N;
}
cin << N
....
```

Updating and conflicts

Once you are happy you have sorted out the problem
`svn resolved bar.c`
will remove all the extra files, it will remove the extra files, so be careful.

Other useful commands

- `svn revert foo.c` : Throws away local changed for foo.c
- `svn diff foo.c -r40` : Displays all the changes to foo.c made since version 40
- `svn status` : Tells which files have changed etc...
- `svn log` : Displays the log messages
- `svn cleanup` : Sort out the mess after an aborted update or commit, most of the time

Other useful commands

- `svn delete foo.c` : Same as delete but tells svn the file has been removed.
- `svn move foo.c ../` : Same as mv but tells the repository about the move.
- `svn copy foo.c foo4.c` : Same as cp, but tells the repository about the copy.
- `svn info` : Gives you lots of information about the repository
- `svn propedit svn:ignore .` : Creates a list of files to ignore in the status mesages
- `svn update -r 40 foo.c` or `svn up -r 40`: Turns back foo.c to version 40 or all files to version 40

Locking, merging and switching

- `svn lock foo.c` : Stops anybody else committing changes to the file
- `svn switch` : Changes the branch a file is in
- `svn merge` : Copies changes between branches

Creating a new branch

1. Create the new branch

```
svn copy  https://svn.mercurydpm.org/SourceCode/Trunk \
https://svn.mercurydpm.org/SourceCode/Branches/myNewFeature/ \
-m "Create a new branches for myNewFeautre"
```

2. Check out the new branch

```
svn checkout \
https://svn.mercurydpm.org/SourceCode/Branches/myNewFeature/ MyOwnBranch
```

Using a branch

- `svn checkout <branchName>` : Checks out the branch
- `svn update` : Updates the branch
- `svn commit` : Commits local changes to the branch
- `svn merge ^\Trunk` : Get updates from the trunk (git pull)

Notes:

1. svn branch do introduce the same two step process of git. Only different is the branches are stored server not client side.
2. If you get

   ```
   svn: E195020: Cannot merge into mixed-revision working copy ...; try updating first
   ```

   If means you local working copy is not update; so do what it say and type `svn up`
3. You should update from trunk regularly; this keeps you in sync with the main development.

Reintegrating a feature branch

1. Make sure you branch is up-to date : `svn up`
2. With the trunk as well : `svn merge ^\Trunk`
3. Commit your final version of your branch :
   `svn commit -m "My feature is done"`
4. Check a clean version of the trunk ”
   `svn checkout https... cleanTrunk`
5. Reintegrate you branch. From the `cleanTrunk` type
   `svn merge ^/Branches/myNewFeature`
6. Check it passes the selftest `make fullTest`
7. Commit the new merged truck (if it passes)
   `svn commit -m "Merge :: Merged NewFeature back into the trunk"`
8. Delete your branch
   `svn delete https://svn.mercurydpm.org/SourceCode/Branches/myNewFeat`

Handy other merge commands

- `svn mergeinfo ^/trunk` : Show which commit have been merged form the trunk
- `svn merge info ^/trunk --show-revs='eligible'` : Show which commit will be merged if you run `svn merge ^/trunk`
- `svn revert . -R` : throw away all changes if the merge went bad
- `svn log -g` : Show the log messages including the ones from the merged branch (it commit is merged)

Undoing commits

- Assume we need to unapply commit 301 to the trunk. To do this type in a working copy of trunk `svn merge ^/trunk -c -301`
  Note
  - Minus, −, in front of 301 means unapply
  - Can be more targeted
  - Merges from one branches to another can individually removed with this command
  - `svn up -r 300` only shows you version 300 it does not revert to this version

  Files can also be resurrected with `svn copy ^/trunk/kernel@299 .`
  - This will copy the kernel directory as of version 299 and place it in the current location
  - Brings all the version history with the file
  - Clearly can be used to resurrect accidentally deleted files

Making a private app which uses trunk

1. Make a new branch

   ```
   svn copy  https://svn.mercurydpm.org/SourceCode/Trunk \
   https://svn.mercurydpm.org/SourceCode/Branches/myNewFeature/ \
   -m "Create a new branches for myNewApp"
   ```

2. Check out the trunk

   ```
   svn checkout https://svn.mercurydpm.org/SourceCode/Trunk
   ```

3. Change one directory to your branch

   ```
   svn switch ^/Branches/myNewFeature/applications applications/
   ```

Note,

- `svn update` will bring updates from trunk to all for applications directory
- `svn commit` likewise will commit change to trunk; except changes to applications which will go to your branch

Making you current checkout version a private branch

1. Make a new branch

```
svn copy  https://svn.mercurydpm.org/SourceCode/Trunk \
https://svn.mercurydpm.org/SourceCode/Branches/myNewBranch/ \
-m "Creating a new branch for my current working version"
```

2. Switch you current working version to be the private branch

```
svn switch ^/Branches/myNewBranch/
```

3. Commit your version

```
svn commit -m "Stashing my broken trunk as I want to back it up"
```

Note

- Very handy trick it you have a non working trunk but you want to stash it some where (better than git stash)
- As long as your new branches is based on the branch you are working the switch will cause no conflicts
- Of course, you can do this to any branch not just the trunk
- Could have used

```
 svn copy myWorkingCopy \
https://svn.mercurydpm.org/SourceCode/Branches/myNewBranch/
```

Conclusion

- Have shown the basics of svn
- Shown the basics of merging
- Much more advance stuff can be done
- Have not covered vendor branches which we will need