
Software Quality Assurance: SOFT3302

Tutorial – Week 1

Objectives

By the end of this tutorial you should have planned your study of this unit for the semester. You should have created a source repository for this unit in your account. This material should also provide an introduction to the basics of source revision control.

Work

There are no scheduled labs in Week 1. You will need to perform these tasks in your own time. It is important that you do these tasks, especially any necessary catch-up reading, this week and not leave them for Week 2.

1. Plan your semester. Make sure you leave time for reading and assignments as well as lectures and tutorials. Do not assume your assignments can all be done in a few days before each deadline. Read the course web site regarding the amount of time you should devote to this subject.
2. Create a source repository in your home directory for use throughout the semester. We will be using subversion this semester for the group project so you might like to familiarize yourself with that if you've not used it before. You may already have such a repository from another course. Create a section in this repository for your SOFT3302-related work. See **Source Control** below for instructions. (If you are already familiar with subversion take this opportunity to refamiliarise yourself with it.)

Use this repository to manage all your work for the rest of this semester.

Source Control

Management of programme source is an important part of any project, large or small. The ability to 'rewind time' to roll-back to or compare with an earlier version is invaluable. This can be done by keeping n copies of your programme directory tree (where n is an every increasing number) but this is intrusive and can lead to weird and wonderful naming conventions (*project*, *project.old*, *project.old.previous*, *project.new*, *project.new.2*, *project.new.old*, and so on...) and either unnecessary duplication of a large amount of unchanged material (if new copies are taken after every simple step) or loss of information (if snapshots are only taken periodically)¹.

The use of a *source revision control* system such as subversion, when set up well, provides an unobtrusive mechanism that solves quite a few programming-related issues, notably

- the management of previous versions;
- the ability to undertake concurrent development with other programmers;
- the ability to undertake experimental development without adversely impacting main-line development while retaining the ability to fold back changes later; and
- the production of patches/diffs for a project.

We will be using subversion for the group project this semester.

¹ The astute reader will realise that this last is a problem related to discipline rather than any particular source control regimen.

The following is a series of short exercises for subversion.

We are going to set up a *repository*, a place where our source control system can manage all its data, and a *sandbox*, a place where we can work before committing things to the repository. **We never work inside the repository, it's a bookkeeping area for the source control system which we never touch directly.**

Setting up a subversion repository and top-level subject directory

Note that subversion does not like running over NFS if one is using Berkeley DB databases. (Rather than using individual files for each file in the repository, subversion has a database containing all files. Changes to the repository become committed checkpoint changes to the database.) One therefore need to use FSFS instead (this is the default in suitably recent versions of subversion). So...

Create a subversion repository in your home directory for use throughout the semester. It is suggested you create with as \$HOME/svn.

```
$ svnadmin create --fs-type=fsfs $HOME/svn
```

It is very important that you *never* touch any of the files in this directory directly.

Use this repository to manage all your work for the rest of this semester. The next exercise creates a directory for SOFT3302 and each week you will need to create a new directory under that for each week. Get familiar with subversion and you can also use it to manage work for your other subjects by creating a new parent directory for each subject.

Create a top level directory for SOFT3302 in the repository (and notice that the repository is not modified directly but through the use of subversion wrapper commands) and export it into your home directory as a sandbox ready for use.² You will be asked to enter a log message: enter 'New subject.' (If subversion barfs saying there is no editor set, set SVN_EDITOR to the path of your preferred editor.)

```
$ svn mkdir file:///HOME/svn/SOFT3302
$ cd $HOME
$ svn checkout file:///HOME/svn/SOFT3302
```

This would be the normal procedure for creating a new top-level directory. You should only need to do this once per subject directory. Weekly directories can be created similarly or by 'adding' a new empty directory into the existing subject-level directory.

If you're familiar with CVS you'll notice that subversion uses URLs to refer to its repository location. This mechanism allows one to refer to a repository accessed via http or ssh more easily.

*The last command creates a new directory called 'SOFT3302' inside your home directory. This is the sandbox directory in which you will do all your work. **You should never work inside the ~/svn directory.***

If you look in the new SOFT3302 you will see a directory called '.svn'. This directory contains all the information subversion needs to do its work on your local copies of files and, just like the main repository, you should never touch any of the files in this directory. In case it's not obvious, you shouldn't delete it either.

² Be aware that a subversion-based project would conventionally have a more complicated structure involving a series of directories under a top-level directory: trunk, branches, and tags. CVS handles these differently.

Adding files to your project

All work related to a project/subject/whatever is done inside your sandbox. Periodically work is committed from the sandbox to the repository.

Once you have created your SOFT3302 directory, create a directory for Week 1 inside it. This will be the normal way you add files and directories to your repository: create the file and then use `svn add` to add it.

```
$ cd $HOME/SOFT3302
$ mkdir Week1
$ svn add Week1
```

Each week you should start with these step: go into your SOFT3302 directory, create a new directory for the current week, add it, change into that directory and then do all work related to that week in that directory.

*When adding file and directories, make sure you always add them explicitly by name, don't use wildcards and especially don't try something like `svn add *` because it could add in other files you don't want to add to the repository like `.o` files and `.pyc` files—this, in general, is a Bad Thing!*

*Note that, by default, subversion will recurse into subdirectories (unlike CVS). Also, subversion will skip directories already under source control so if you need to force it to look 'lower down' when files have been added all over a tree use `svn add * --force`. Don't use this until you know what you're doing.*

Committing work to the repository

You now have a Week 1 directory. Go to it and create a simple 'hello world' programme (a programme which simply prints 'Hello, world.')

 in a language of your choice and add it to the repository and commit it.

```
$ cd $HOME/SOFT3302/Week1
$ vi hello.py
$ svn add hello.py
$ svn commit hello.py
```

You will be asked to create a log message to describe your programme. Write something descriptive. **Never EVER use comments such as 'fixed bug' or 'made some changes'**. Your programme will be committed as some integer number in subversion (remember this number as `svn_r1` and let's say for the sake of this tutorial it was 4).

Don't worry about these version numbers, they are internal to subversion and are useful for determining the different revisions of your file. It's not unusual in subversion to see something like version 1156 of a file.

If you are used to CVS you will notice that CVS and subversion track revision numbers quite differently. Under subversion the version number will be an integer. Again, since these are for internal use only it doesn't matter. Anything of specific interest can be given a symbolic name by copying a specific revision to another part of the file hierarchy. This is quite an efficient operation as the subversion repository will not store a duplicate copy, merely a reference to the previous revision.

Change your programme to say 'Goodbye, everyone.' rather than 'Hello, world.' and commit the change. Make sure you write a suitable descriptive log message, more than just 'made changes.' These messages are very important.

```
$ vi hello.py
$ svn commit
```

This should create a version ‘an integer larger than previous integer’ ☺ under subversion (remember this number as *svn_r2* but again imagine it was 7).

Comparing source changes

Have a look at the differences between the first version (whatever you wrote down as *svn_r1*) and the second version (*svn_r2*). Substitute the numbers you wrote down for 4 and 7 in the subversion example below.

```
$ svn diff -r 4:7 hello.py
```

This is one of the important benefits of a source control system: the ability to look at (and roll back to if necessary) old versions of a file. You should get into the habit of making short edit-test-commit cycles changing or adding one feature at a time rather than doing a whole day’s worth of work before committing.

End matter

Well, that’s basically it as an introduction. Continue this cycle:

- Add/modify files
- Test files
- Commit working files to repository

(We didn’t cover the second step but you do test your files, don’t you? And you would never commit unworking code to a repository, would you?)

Keep this cycle short as it gives a finer grain if you have to go back and pick out a mistake/change that happened a long time ago. (Think taking out a small 10-minute change committed on its own rather than having to dig through an entire day’s or week’s worth of changes to find the same few lines).