Introduction to Version Control

- Practical -

15 June 2015

Before starting this practical

We recommend that before working through this practical you watch the first ARCHER virtual tutorial on version control as well as the accompanying live demonstration of Subversion (SVN), available from the links provided below.

Virtual Tutorial - Introduction to Version Control (Part 1):
https://www.youtube.com/watch?v=S2iddvouVyw

Live demonstration of SVN:
https://www.youtube.com/watch?v=V53NGLpuI-A

Introduction and Aims

The aim of this practical is to allow you to gain hands-on experience creating a repository and using a version control system to manage your files. You will learn how to branch, merge, and resolve conflicts. You will do this using Subversion (SVN) – the version control system demonstrated during the first virtual tutorial. The practical will teach you how to start using this tool and how to deal with some of the practicalities surrounding version control.

Practical

Version control from the command line

In this practical we will be using SVN from the command line. Although a web-based interface or a standalone application with a graphical interface can be convenient it may hide the direct commands issued to the version control system as well as any error messages it returns. By using these tools directly from the command line we get a clear picture of how they work, how to interpret any errors, and what to do when something unexpected happens. We also learn how to access the built-in help system to figure out or simply remind us how to accomplish a given task.

If your own machine does not already have SVN installed (e.g. if you are using Windows) or if you encounter any problems with the versions installed on your machine you can log on to ARCHER and perform the practical there.

Example content used in this practical

For the purposes of this practical we want to use example content that is easily understood and that helps illustrate the principles and practice of version control. Instead of source code the example content that we will place under version control consists of culinary recipes. This is not so arbitrary or frivolous because as well as being easily understood there are analogies between recipes and code that make it a useful choice. Cooking instructions are like an algorithm, and the ingredients used are like variables. We will split each recipe over two separate files:

1. `recipe.instructions`: cooking instructions

2. `recipe.ingredients`: list of ingredients

For a given recipe every ingredient used in the instructions must appear in the corresponding list of ingredients, just as the variables used in a piece of code must be declared. Matching instructions and ingredients files for a recipe should therefore be committed at the same together to the repository, otherwise a person trying to use that version of the recipe will not be able to cook it due to missing ingredients or be wastefully stuck with unused ingredients. This is analogous to not being able to compile a version of a set of source code files because not all dependencies are met, or receiving warnings because declared variables go unused. Hence introducing this simple dependence between recipe files makes our use of version control more representative of managing real code.

Create a repository

Imagine you are writing a cookbook and plan to go through a process of experimentation, trying different variations for each recipe. At the end of this process you want to be able to choose which versions of recipes to publish. The first step in this practical is to create a repository to keep track of the different versions of recipes.

In the live demonstration we dealt with an existing Subversion repository, thereby neatly circumventing the question of how one creates one in the first place. If you type `svn help` to show a list of available Subversion commands you will not see any command that looks like it can be used to do this. In fact there exists a separate utility called `svnadmin` that is used to create and administer Subversion repositories, however using `svnadmin` requires more detailed explanation than we want to provide here, and would take more effort to use than we think is reasonable for a first-time SVN user.

If your own institution provides Subversion hosting this is the perfect opportunity to sign up and familiarise yourself with the service and use it to create a repository (call it "recipes") for this practical, noting down the URL at which your repository is accessible. Otherwise for the purposes of this practical you can make use of a website that hosts Subversion repositories called Assembla. You will need to visit http://www.assembla.com and create a free trial account. When prompted to "Name your space" during the signup process choose "uname-recipes" (replacing "uname" with your username) – this will be the name of your repository. Choose "Add a Subversion repository". Your repository will be available at https://subversion.assembla.com/svn/uname-recipes/. If at any point you find you have lost control or understanding of the state of the repository and want to start the practical again with a clean repository you can always delete it on the website and create a new one.

Check out a working copy

Next, open a terminal window on your machine or on ARCHER and navigate to a directory where you are happy for a local working copy of the repository to live. Check out the repository by doing

```
svn checkout --username uname
https://subversion.assembla.com/svn/uname-recipes/
```

(one line), replacing `uname` with your Assembla username. If you are asked to accept a server certificate choose "p" for "(p)ermanent", and enter your Assembla password. You should see the following notification of directories and files that were added (A) to your working copy:

```
A     uname-recipes/tags
A     uname-recipes/trunk
A     uname-recipes/branches
A     uname-recipes/readme.textile
Checked out revision 1.
```

The `trunk` directory is typically where one begins adding content. Playing the role of a main, or master branch, it often ends up containing the most important, most permanent or most accepted versions of content. It typically functions as the primary content from which other branches, usually located in the `branches` directory, are derived. In the context of software development the trunk usually contains the source code for the full stable version of the software, while other branches are used to develop and test new features, some of which are later integrated back into trunk by merging.

It is worth mentioning that although SVN repositories are initialised by default to contain `trunk`, `branches` and `tags` directories, this is purely a convention. It is not strictly necessary and in no way enforced, and you are free in principle to create a different repository structure. For the purpose of this practical we will stick with the convention.

Enter the `trunk` directory and, using your preferred text editor, create two files: `guacamole.ingredients` and `guacamole.instructions`. Edit these files to contain the basic recipe given here:

`guacamole.ingredients`:

```
Avocados
```

`guacamole.instructions`:

```
Mash avocados
```

We would like to record our basic recipe in the repository. Although we've placed these files within a folder in the working copy of the repository we still need to explicitly tell SVN that we want to place these files under version control, i.e. that we want to add them to the repository:

```
svn add guacamole.ingredients guacamole.instructions
A       guacamole.ingredients
A       guacamole.instructions
```

Looking at `svn help add` reveals that we still need to commit these files before they are actually recorded in the repository:

```
svn help add
add: Put files and directories under version control, scheduling
them for addition to repository. They will be added in next
commit.
```

When giving the command to commit these files we must include a commit message using the `–m` option followed by a string. We may need to specify our Assembla username using the `--username` option, though SVN should remember this from before.

```
svn commit –m "Added basic guacamole recipe"
Adding         guacamole.ingredients
Adding         guacamole.instructions
Transmitting file data ..
Committed revision 2.
```

A note about committing: by default `svn commit` will commit all changes to files already under version control located in the current working directory and its subdirectories. What these changes are can be shown using `svn status`. However you are free to select only the changes to particular files to be committed by specifying those files after the commit command.

If we now update our working copy:

```
svn update
Updating '.':
At revision 2.
```

We will see our commit shown in the log:

```
svn log
------------------------------------------------------------------------
r2 | uname | 2015-06-16 06:46:02 +0100 (Tue, 16 Jun 2015) | 1 line
```

```
Added basic guacamole recipe
------------------------------------------------------------------------
r1 | www-data | 2014-08-26 21:22:14 +0100 (Tue, 26 Aug 2014) | 1 line

Automatically created readme.textile and /trunk, /branches, /tags
directories. We recommend you to put all your code there.
------------------------------------------------------------------------
```

We quickly realise that we can improve on our basic recipe by adding the following lines to each file:

```
guacamole.ingredients:

Avocados
Lime juice
Salt

guacamole.instructions:

Mash avocados
Stir through lime juice
Stir through salt
```

Looking at the status:

```
svn status
M       guacamole.ingredients
M       guacamole.instructions
```

We can see that SVN knows we've made these changes ("M" stands for modified) so a simple commit will record them and bring the repository to revision number 3:

```
svn commit –m "Improved basic guacamole recipe by adding salt and
lime juice"
```

We can examine our most recent commit in the log (using --verbose to provide extra information):

```
svn log –l 1 --verbose
------------------------------------------------------------------------
r3 | uname | 2015-06-16 06:51:49 +0100 (Tue, 16 Jun 2015) | 1 line
Changed paths:
   M /trunk/guacamole.ingredients
   M /trunk/guacamole.instructions

Improved basic guacamole recipe by adding salt and lime juice
------------------------------------------------------------------------
```

Branching

Starting with this improved basic recipe we now want to experiment with some additions and develop the recipe in a particular direction – perhaps to explore what we think are some authentic flavours. However we're not sure whether we're going to like this new recipe and hence whether we want to keep it at all and want it to be recorded in trunk for eventual publication in the cookbook. So we create a branch called `authentic` consisting of a copy of the basic guacamole recipe stored in `trunk`, and place it in the `branches` directory. In SVN this is done using the `copy` command, for which there are several usage options:

```
svn help copy
copy (cp): Copy files and directories in a working copy or
repository.
usage: copy SRC[@REV]... DST

  SRC and DST can each be either a working copy (WC) path or URL:
    WC  -> WC:   copy and schedule for addition (with history)
    WC  -> URL:  immediately commit a copy of WC to URL
    URL -> WC:   check out URL into WC, schedule for addition
    URL -> URL: complete server-side copy; used to branch and tag
```

Using the fourth option, i.e. giving repository URLs for both source (SRC) and destination (DST), is recommended for branching. This tells SVN to perform a copy directly in the repository, i.e. on the server where the repository is located, rather than in our working copy. Instead of specifying the entire Assembla URL we can use the caret symbol "^". This is an SVN shorthand for the URL of the repository root, in this case https://subversion.assembla.com/svn/aproeme-recipes/.

```
svn copy ^/trunk ^/branches/authentic -m "Creating authentic
guacamole branch"
```

Alternatively we could have chosen both source and destination to be in our local working copy:

```
svn copy trunk branches/authentic
```

However this would have required a separate commit afterwards to record the change, and more importantly the URL → URL option is faster especially when large numbers of files are concerned.

Note that if `trunk` had also contained other recipes we would have issued a copy command just for the guacamole files instead of for all of `trunk`, although in another situation we may actually want to create a branch containing copies of *all* recipes in `trunk`, for example a vegetarian branch containing meat-free versions of all recipes.

Before proceeding we should again bring our working copy up to date with the repository. Like many SVN commands the `update` command only operates on the current working directory and its subdirectories. In order to update the working copy with the changes made to the `branches` directory in the repository this command should therefore be issued in or above the `branches` directory in the working copy – if it is run in `trunk` it will not actually update our working copy with the new branch. The last entry in the log shows that the `authentic` directory was created based on the state of `trunk` in revision 3:

```
svn log -l 1 --verbose
------------------------------------------------------------------------
r4 | uname | 2015-06-16 07:05:17 +0100 (Tue, 16 Jun 2015) | 1 line
Changed paths:
   A /branches/authentic (from /trunk:3)

Creating authentic guacamole branch
------------------------------------------------------------------------
```

Now enter `branches/authentic` and modify the guacamole recipe so it reads:

`guacamole.ingredients`:

```
Avocados
Salt
Lime juice
Onion
Tomato
Coriander
```

`guacamole.instructions`:

```
Mash avocados
Stir through salt
Stir through lime juice
Add chopped onion
Add chopped tomato
Add chopped coriander
```

Commit these changes with an appropriate commit message.

Merging

We like this recipe so much that we plan on including it in our cookbook. The final cookbook recipes will all be taken from `trunk`, so we want to merge the `authentic` guacamole recipe back into `trunk`. From looking at `svn help merge` it is clear there are a number of ways to do this. Since the `authentic` branch only contains changes to the guacamole recipe, the easiest way is simply to merge the entire branch into trunk:

```
svn merge ^/branches/authentic trunk
```

Although the source is the URL of the checked in version of `authentic` in the repository, unlike for branching the destination is `trunk` in our working copy. The reason for this is that merging can lead to conflicts, which need to be resolved. This is tackled within our working copy, and once done the resolved version can be committed to the repository.

The result of the above merge command is

```
--- Merging r4 through r5 into 'trunk':
U    trunk/guacamole.instructions
U    trunk/guacamole.ingredients
--- Recording mergeinfo for merge of r4 through r6 into 'trunk':
 U   trunk
```

This tells us that the two files in `trunk` as well as some metadata about how the repository has changed (`mergeinfo`) have been updated (U). If we examine the status of trunk:

```
svn status trunk
 M      trunk
M       trunk/guacamole.ingredients
M       trunk/guacamole.instructions
```

we see that the two files as well as the `trunk` directory itself are marked as modified (M). To record the result of our merge operation in the repository you should now commit these changes.

Resolving conflicts

Finally we are going to examine how to resolve a conflict. This can happen if two different authors edit the same file, or if you yourself have made modifications to the same file in two different locations.

To explore the latter situation, open a second terminal window on your machine and navigate to a different directory than the one where you originally checked out a working copy of the repository. Now check out the same repository again. Go to `trunk` and modify the first line of `guacamole.instructions` to read:

```
Mash avocados roughly – leave chunks
```

Commit this change. Now go back to your original terminal window and the location of the first working copy you checked out, and change the same first line in `guacamole.instructions` in `trunk` to read:

```
Mash avocados finely to a smooth paste
```

Now attempt to commit this. You will be told:

```
Transmitting file data .svn: E155011: Commit failed (details follow):
svn: E155011: File '~/uname-recipes/trunk/guacamole.instructions' is
out of date
svn: E170004: File '/trunk/guacamole.instructions' is out of date
```

This is because you committed changes to the same file, hence your working copy is out of date with respect to the repository. You will have to update your working copy before you can commit anything. When you update SVN attempts to merge the latest version of `guacamole.instructions` in the repository with the version in your working copy, leading to a conflict (marked with file status C) because the first line differs in the two versions:

```
svn update
Updating '.':
C    guacamole.instructions
Updated to revision 8.
Conflict discovered in file 'guacamole.instructions'.
Select: (p) postpone, (df) show diff, (e) edit file, (m) merge,
        (mc) my side of conflict, (tc) their side of conflict,
        (s) show all options:
```

SVN is asking you how to resolve this conflict. You could start by responding with `s` to show more information about the options. You can show what the difference is between your working copy of the file and the most recent version being fetched from the repository in the update by responding with `df`.

You may like to explore what happens for different options. If you like the version of the file you checked into the repository first (rough chunky avocados) you may want to simply accept it by responding with `tc`, this overwrites your working copy of the file with the checked-in version. Alternatively if you insist your working copy version (smooth avocados) is better you can choose to keep it by responding with `mc`, optionally followed by a commit that records your version in the repository. You may choose to edit the file (e) to put in some kind of compromise choice on how to mash the avocado, followed by a commit of that version. You can also postpone the decision and come back to it later.

As you can see the SVN terminology ("my", "their") reflects the situation of multiple authors editing the same file. In this case you should probably communicate with your co-author / collaborator to come to a decision, and then implement it using the version control system.

You are encouraged to explore these options and to use `svn help` to better understand the commands used so far as well as to explore new commands. Online documentation and guidance about using SVN can be found at http://svnbook.red-bean.com/