SVN for FTP Recidivists

Jonathan Guthrie, xServe Ltd; Wellington, New Zealand

The challenge

Imagine the following scenarios.

> You work in a workgroup of developers and you finish some mods on a document and FTP it up to the server, press the appropriate "ok to replace" button, and half an hour later a colleague sends an email to the group angry because his changes have just been overwritten…
>
> You've spent the last few hours writing a script to manage and organize file uploads. The code works, you're happy with your code and you test it. Later that day you open it again and make a quick change to a few variables and close the file. A couple of days later you find that file that you thought you had sorted has an error, and yet you were positive it worked before, but can't remember what it was you changed. You don't really have any way of reverting back to that time when you knew 100% that it worked…
>
> You've worked on a complex page the whole day, and then you find you have accidentally saved over the top, and it's all gone…
>
> You have been working on changes to a client's site for the last week and now you want to FTP them up to the live server – however you don't know exactly which files were changed, and you really don't want to FTP the whole project up afresh because it's huge and you didn't copy down all the files. You have to comb through the whole site comparing last mod dates, hoping you don't miss anything…

Some of these scenarios sound pretty stupid, some achingly familiar, and even if you're still using FTP, AFP, or SMB you're still at risk of needing to use another much ruder acronym.

So if you are currently a frequent user of one of the above TLA's I want to introduce you to an alternative, one that I personally find is better; SVN, or to use it's real name, Subversion.

Why am I so 'sold' on Subversion? Because it gives me maximum earning power, with minimum mess ups, and it makes the most of team work. Making team work, effective is a valuable thing as I work in the opposite time-zone to most of my fellow developers. Lets look at how this is done.

Enter the world of Subversion

Subversion is an implementation of a Version Control System (also known as Source Control, Source Code Management or Revision Control).

Revision Control is defined in Wikipedia as:

> … the management of multiple revisions of the same unit of information. It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code, art resources such as blueprints or electronic models, and other projects that may be worked on by a team of people.

So what does that mean for us practically?

It means that if you are working on a document that a colleague is also working on, SVN will alert you to conflicting changes and you have several ways to resolve the conflicts.

If you have broken a script or simply killed it by overwriting it, you can either roll it back to any previously committed version, or compare your existing document to a specific prior version and selectively incorporate differences.

But wait! There's more!

How about splitting off a whole new branch of code? By using the SVN's branching feature you can create a whole new project based on an existing one.

Do you like to re-use common chunks of code, like FCK Editor or Custom Tags and Types? No problem: SVN Externals are like directory aliases between projects.

Want to put marker milestones in place so that you can deploy specific point revisions of a project? Use the Tags feature to create a snapshot of your project.

Want to be able to have multiple servers all working off the latest version of the trunk or branch, or all on the same tag? It's easy to script servers to pull the right data from a central SVN server.

Some practical how-to's

So lets look at some practical examples.
For this presentation I will be using Eclipse fitted out with Lasso Studio for Eclipse, and Subclipse, a SVN module for Eclipse that integrates SVN function into the Eclipse workflow experience.
Typically, developers working with FTP often write their code on their workstation and upload this to the server to test. Developers using a file share protocol often work directly on the server. This makes sense to some degree in that there's only one lasso setup, and one database setup.
Working with SVN requires a slight mindshift away from central share code access towards a more independent working environment. Tom Wiebe educated me in the term "sandbox" when explaining this concept to me - and this again is found well described in Wikiedia.

> A sandbox is a testing (or virtual) environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including web development and revision control…

So, your own sandbox would typically mean that you have locally:

- Lasso Developer or Lasso Studio for Eclipse
- A datasource such as MySQL or similar
- Apache
- A hosts file modified to suit
- Eclipse / Subclipse

Your setup for the Lasso site will mirror the settings required for deploying the site on the production server. Your local datasource will have a copy of the database(s) used in the project. You can use a subset of the data if you so wish – there's no sense in having your projects' full database if size and/or confidentiality is an issue. It makes sense to mirror any custom or special configuration directives from the live sites' Apache config. Remember, the whole point of this exercise is to duplicate the functionality and the behavior of the live site as much as is practically possible.
However, this is where the full force of the concept of a sandbox comes in: the best thing is that even if you make a mess of your own sandbox, you don't impact the live servers and you don't mess up your colleagues' sandboxes. Most importantly, if you've made a really big mess you can delete your database and restore from the snapshot and delete your project directory and check it out afresh from the SVN server!
This paper is not going to cover the setup of Lasso Developer/Studio, datasource setup, Apache vhost setup or even the install of the SVN server itself – instead this paper is now going on to the business end - some examples of how to manage your code in the Eclipse setup I described earlier.

Working with production and staging servers

Releases to production servers should always be tested thoroughly. However it is impractical, from both a time and cost perspective, to expect development and repository commits to be suspended until that testing is complete and the code is updated on the live servers.
That is why we have **"tags"**.
Tags are a way of marking a specific point in the development of a project – 2.3.4.02 as example. The leading "2" would be the major revision number, the 3 the minor revision, or cycle of releases. The 4 could be the release you're planning on deploying, and the 02 would then be the bug fix revision based on the testing to harden the code prior to deployment.
So you tag a release as 2.3.4.00, test, and discover some minor bugs you need to squash, you fix them, re-tag as 2.3.4.01. You continue until it's solid.
Say 2.3.4.02 is your solid build – that's the one you should export to the production server. You should either issue the command from the terminal, or use a GUI client like SvnX to do this. There are a couple of options here as well. You can either "switch" the link that the production server files are attached to in SVN (ie from tagged release 2.3.3.05 to 2.3.4.02), or you do a new export and change the DocumentRoot directive in your apache virtual host file.
If you do an export the version info is not retained as it strips out all SVN specific info, whereas if you do a checkout, version info can be retrieved using SVN and urgent updates to that tag can still be applied if required. Either way, where infrequent staged releases are deployed, it is useful on the server side to visually identify which version or tag has been used.

To define the Repository:

Select *Window -> Open Perspective -> SVN Repository Exploring* (You may need to select "other" menu item and choose from the list)

Click the "Add repository" button as below:

Enter the location of the repository, Click finish. The location will be a URL similar to "svn://myserver.com/usr/local/svn/repository/"

You will see the repository list as below.

Click on the triangle to expand, you should see an authentication dialog like the one below appear:

Enter your username and password as supplied, click OK.
The repository list will be fetched from the server and returned in hierarchical form.

Checking out a new project from the repository

To begin working on the project, you first need to check it out from the repository.
Note: If you are starting a new project instead, you will need to see the *"Creating new projects in the repository"* instructions in the following section.
Right-click on the directory you wish to check out as shown below, and choose "Checkout…"

Click "Finish" to use the "New Project Wizard". You will then be presented with the following screen:

Choose "Lasso Project" and click "Next"

Choose an appropriate name, and if you don't want the project stored in the default location uncheck "Use default location" and navigate to your new directory under "Browse…"

On clicking "Finish" your project will check out from the server.
Your project is now "connected" to the SVN server, and when you make changes Eclipse will show a different icon for the file or directory depending on it's status.
It will signify changes, additions, conflicts.
Note: The console pane is also a worthwhile addition to your perspective. It will show you what's been updated, and where potential problems lie.

Working with Eclipse and SVN

Committing

When you make a change to a file the icon changes as seen in the image below.

The * means it's a local change, and should be committed to SVN.
You don't need to commit a change immediately to the repository but you should get in the habit of doing so regularly. Typically this means committing the changes at each "stage" completion, or when a bug/issue has been resolved involving the files concerned.
At the minimum it is best practice to commit your changes at the end of each day so other developers do not get too many chances for their work to conflict with yours.
When committing, you choose how high or wide to go: you can commit a specific file, a selection of files, the set of changed files in a specific directory, or all changes within the project.

It is important to make sure that you comment your changes on commit. These comments are crucial to other developers (and often yourself!) knowing what was changed and when, and by whom. When needing a rollback or restore, these comments will also aid you in knowing a "last good state".

Updating other users' changes to your own sandbox

At the beginning of each work session it is advisable to do an update of the entire project to ensure you are incorporating other developers' recent work. This is crucial to help you avoid conflicts (situation where more than one developer works on a given file). If heavy development is underway you may even decide to update your sandbox more frequently to be comfortable that the alterations you are making to the codebase work in harmony with the other concurrent work.

Creating a branch

Creating a branch is an easy way to either create an entirely new iteration of a site founded on a "base code", or to create a version of the codebase with which you'd like to do major surgery to without affecting the normal run of bug-fixes and updates to the trunk. This allows parallel development by either yourself or multiple developers.
Select *Window -> Open Perspective -> SVN Repository Exploring*
Right click on the directory you'd like to copy/branch/tag

Select Branch/Tag…

Enter the destination URL, or navigate to it using the "Browse" button. Enter a comment and click OK.
Your branch is done and ready to check out and start using.

Creating new projects in the repository

When you have code you wish to commit afresh to a repository that you have already defined, right click on the project in the resource view navigator, select Team > Share Project…

Select your SVN repository location, specify the folder name, click next and enter an initial commit comment.
The project is then committed and attached locally to your SVN repository.

Some final thoughts

It is good practice to do an update from the repository each morning at least – or more often if frequent work from multiple developers is taking place. This will help minimize conflicts that need to be resolved.
The larger the projects and the tougher the timeframes, the more you and your colleagues will benefit from using this development tool. Actively investing in learning these skills on smaller projects will set you up to win when you or your company lands 'the big one'.
This tool also makes re-use of your code much more efficient, and once in place it is much more easily maintained. This in turn gives a serious boost to your ability to output new sites cleanly and in record time.
Maximum earning power, minimum mess ups, and making the most of team work.