Using Bazaar Branches

Subversion is not needed for working with Subversion repositories

Just because a central repository is a Subversion repository, it doesn't mean you have to use Subversion to work with it. Subversion requires connection between client and server for most of the operations relating to commit and history, disconnected working is definitely not well supported. Moreover, Subversion does not support branching and merging well; creating experimental branches, merging results, etc. are hard with Subversion.

Distributed version control systems are on the rise. I am particularly interested in Bazaar as it is straightforward to use and supports the way I want to work. For a general discussion of Bazaar please look at http://www.bazaar-vcs.org. Here I am going to cover only Gant (and Groovy) specifics.

Bazaar works with the idea of a branch. A branch is a copy of the entire history. Usually a branch has a working tree. So whereas a Subversion checkout has the working tree and the checkout tree (allowing diffs and reverts to be entirely local operations), a Bazaar branch has the entire history and the working tree (allowing all operations to be entirely local). Bazaar branches can be standalone (the normal case for distributed working) or they can be bound to another branch so that operations on the local branch are also carried out on the branch that is bound to.

There is a plugin to Bazaar (bzr-svn) that allows a Subversion repository to be treated as though it contains a collection of Bazaar branches.

Put all this together and it means that you can take a checkout of a branch in a Subversion repository using Bazaar: a checkout is just a branch that is bound. Of course a checkout is only really useful to someone with write permission to the bound branch. For people that do not have write access to the central repository, there is no point in taking a checkout, but you can take a branch. You get all the capabilities of a version control system but without having to be a committer to the project. This is what distributed version control is all about, everyone can undertake recorded evolution of project source. Committers can then commit directly, non committers can submit patches. Subversion just does not allow this width of contribution.

Branching out

A Bazaar branch of the Gant repository trunk is taken with the command:

> *bzr branch svn+http://svn.codehaus.org/gant/gant/trunk Gant_Trunk*

where I have chose to name the directory in which the branch appears Gant_Trunk, you can use whatever name you want. If you are more into GUI tools, then on Windows there is TortoiseBzr and on GTK-based systems, Olive-GTK.

For people with write access to the Gant repository, then a checkout is probably what you want:

> *bzr co svn+https://svn.codehaus.org/gant/gant/trunk Gant_Trunk*

This is just a branch that is bound to the Subversion store.

In the directory Gant_Trunk is a working tree ready for working on and a directory .bzr that contains all the information about the branch, including who the parent was and whether the branch is bounf or not.

> *bzr info*

will tell you the information you need to know about these things.

> *bzr log*

will give a full listing of the entire recorded history of the project as known in this branch.

If I make some changes to the working tree, then I can use

> *bzr status*

to find out information about the branch, and

> *bzr diff*

to find out differences between the working tree and the last commited in the branch.

All this should seem very comfortable to a Subversion user. The difference come when changes are made to the branch.

> *bzr commit*

will commit changes from the working tree to the branch. Unless the branch is bound, commit is a completely local operation. If the branch is bound then as well as the local operation, a commit to the non-local branch will be made. In the case of Gant, a commit to a checkout (aka bound branch) will cause a commit to the Subversion repository as well as the local branch.

All of the commands can take parameters to restrict operation to just the named file, not much different from Subversion really 

If at any time you can't think what to so, Bazaar has build in help, so for example:

> *bzr help commit*

So why use Bazaar instead fo Subversion? Two things:

1. Disconnected committing.
2. Ability to branch further (for experimentation).

Disconnected working

You are not always connected to the Internet when you want to make a commit. This is the biggest problem with Subversion - no connection, no commit. With Bazaar you can commit to the local repository even with a bound branch.

> *bzr commit --local*

will commit only to the local branch Of course if the branch is not bound then this doesn't change the behaviour. For bound branches though it performs the commit locally but does not attempt to perform the commit to the branch bound to.

When it is time to commit to the branch bound to, we do:
.bq bzr update
to get all the commit information sorted out and then:
.bq bzr commit
all the originally locally-only committed changes are committed as individual changesets so that none of the commit history is lost.

For situations where extended periods of disconnected working are required, or it is desired to batch up a number of commits, the the branch can be unbound:

> *bzr unbind*

the information about the branch bound to is not lost. So after various commits (we don't need the --local now as the branch is unbound), when we want to commit to the other branch:

> *bzr bind*

> *bzr update*

> *bzr commit*

Branching further

Just branching

Subversion has repository and checkouts. Bazaar has branches. In general all branches are equal, but yes some branches are more equal than others. If we have a branch then we can branch from it to create another branch that has the same shared history. So say we:

> *bzr branch Gant_Trunk Gant_Experiment*

then we have two branches which are complete in themselves. We could even just do:
bq cp -rp Gant_Trunk Gant_Experiment
but by using a branch command it is likely that the most compact representation of the branch is created.

Making branches more efficient - shared repositories

When you know you are going to make branches of branches, it is almost certain you want to use a shared repository so as to make disk use as efficient as possible.

For Gant we can create a shared repository:

> *bzr init-repository --rich-root-pack Gant*

then we can populate it with a checkout:

> *cd Gant*

> *bzr co svn+https://svn.codehaus.org/gant/gant/trunk Trunk*

so we have a directory Gant which is a shared repository and in that a directory Trunk which is a checkout (aka bound branch). Trunk is a branch but instead of all the history being helpd in Trunk/.bzr, it is held in Gant/.bzr so that it is available for sharing by other branches. The other branches must be in the same Gant directory of course.

So:

> *bzr branch Trunk Experiment*

creates a new branch Experiment which is a copy of Trunk but which shares all history up to the point of branching. This makes branches very cheap and so we do not have to be afraid to make them. This means it is easy to try experimentation.

Merging the results

How to get the results of experimentation back into Trunk? This is where:

> *bzr merge*

comes in handy.

<unfinished>