Nested items

Introducing nested items

For many large projects, it is often useful to incorporate libraries maintained elsewhere or to construct them from multiple subprojects. While it is easy for a single user to setup a particular layout of multiple branches by hand, the different branches really need to be linked together if others are to reproduce the desired layout, and if the relationships are going to be managed over time.

Bazaar has good support for building and managing external libraries and subprojects via a feature known as *nested items*. In particular, nearly all of Bazaar's commonly used commands understand nested items and Do The Right Thing as explained below. In general, nested items are stored in directories of a *containing branch*. The relationship is hierarchical: the containing branch knows about its nested items but nested items are unaware of the branch (or branches) containing them.

At the moment, *nested branches* are the only type of nested item supported though *nested files* may be supported in the future. Nested branches may contain other nested branches as required.

Note: This feature requires a recent branch format such as `2.0` or later.

Nesting an external project

To link an external project into a branch, use the `branch` command with the `--nested` option like this:

```
bzr branch --nested SOURCE-URL TARGET-DIR
```

For example, assuming you already have a `src/lib` directory where libraries are kept:

```
bzr branch --nested http://example.com/xmlsaxlib src/lib/sax
```

This will create a nested branch in the `src/lib/sax` directory, join it into the containing branch and save the source location.

If you now run `bzr status`, it will show the nested branch as uncommitted changes like this:

```
+  src/lib/sax
+  src/lib/sax/README
+  src/lib/sax/parser.py
...
```

To record this change, use the `commit` command as you normally would:

```
bzr commit -m "added SAX parsing library"
```

Note that Bazaar stores the tip revision of each nested branch. This is an important feature in that it's then easy to reproduce the exact combination of libraries used for historical revisions. It also means that other developers pulling or merging your changes will get nested branches created for them at the right revisions of each.

Refreshing a nested branch

As bugs are fixed and enhancements are made to nested projects, you will want to update the version being used. To do this, `pull` the latest version of the nested branch. For example:

```
bzr pull src/lib/sax
```

If the latest revision is too unstable, you can always use the `-r` option on the `pull` command to nominate a particular revision or tag.

Now that you have the required version of the code, you can make any required adjustments (e.g. API changes), run your automated tests and commit something like this:

```
view src/lib/sax/README
(hack, hack, hack)
make test
bzr commit -m "upgraded SAX library to version 2.1.3"
```

Changing a nested branch

As well as keeping track of which revisions of external libraries are used over time, one of the reasons for nesting projects is to make minor changes. You may want to do this in order to fix and track particular bugs you need addressed. In other cases, you may want to make various local enhancements that aren't valuable outside the context of your project.

As support for nested branches is integrated into most commonly used commands, this is actually quite easy to do: simply make the change to the required files as you normally would! For example:

```
edit src/lib/sax/parser.py
bzr commit -m "fix bug #42 in sax parser"
```

Note that Bazaar is smart enough to recurse by default into nested branches, commit changes there, and commit the new nested branch tips in the current branch. Both commits get the same commit message.

If you want to only commit the change to a nested branch for now, you can change into the nested branch before running commit like this:

```
cd src/lib/sax
bzr commit -m "fix bug #42 in sax parser"
```

Alternatively, you can use a selective commit like this:

```
bzr commit -m "fix bug #42 in sax parser" src/lib/sax
```

Reviewing nested branch changes

Just like `commit`, the `status` and `diff` commands implicitly recurse into nested branches. In the case of `status`, it shows both the nested branch as having a pending change as well as the items within it that have changed. For example:

```
M src/lib/sax
M src/lib/sax/parser.py
```

Once again, if you change into a nested branch though, `status` and `diff` will operate just on that branch and not recurse upwards by default.

Browsing nested branch history

As nested branches have their own history, the `log` command shows just the history of the containing branch. To see the history for a nested branch, nominate the branch explicitly like this:

```
bzr log src/lib/sax
```

Note however that `log -v` and `log -p` on the containing branch will show what files in nested branches were changed in each revision.

Undoing nested branch changes

While committing in a containing branch will commit in nested branches by default, `uncommit` works in the opposite way, i.e. it recurses up by default, not down. As explained above, if you make a change to a nested branch like this:

```
edit src/lib/sax/parser.py
bzr commit -m "fix bug #42 in sax parser"
```

then two commits are actually done: one to the nested branch and one to the current branch. The way to undo that commit pair is:

```
bzr uncommit src/lib/sax
```

If you `uncommit` the current branch, then just that commit is undone and the commit to the nested branch is left intact. The reason for this behaviour is simple: Bazaar doesn't know whether the commits were done as multiple steps or not and whether you want one or both commits undone. In comparison, it doesn't have a choice if you uncommit the change made to the nested branch. In that case, it *must* rollback the higher level commit because the referenced tip revision no longer (logically) exists.

Splitting out a project

If you already have a large project and wish to partition it into reusable subprojects, use the `split` command. This takes an existing directory and makes it a separate branch. For example, imagine you have a directory holding UI widgets that another project would like to leverage. You can make it a separate branch like this:

```
bzr split src/uiwidgets
```

To make the new project available to others, push it to a shared location like this:

```
cd src/uiwidgets
bzr push bzr://example.com/uiwidgets
```

You also need to link it back into the original project as a nested branch using the `join` command like this (assuming the current directory is `src/uiwidgets`):

```
bzr join --nested .
bzr commit -m "uiwidgets is now a nested project"
```

Similar to `branch --nested`, `join --nested` joins the nominated directory (which must hold a branch) into the containing branch and saves the branch's public location, if any, as its location.

Updating nested branch locations

As well as storing the tip of each nested revision, Bazaar keeps track of the location of nested branches. Unlike tip revisions though, nested branch locations are **not** tracked over time. While new locations are propagated by the expected commands (e.g. `pull`, `merge`, `update`, `push`, `commit` on a bound branch), any existing values are left untouched by `merge` etc. This behaviour is explicit: it simplifies bootstrapping of locations while allowing users to provide custom locations that refer to local mirrors.

The `nested` command is used to list, view and manage nested branch locations. To view all locations:

```
bzr nested
```

To view the location of a single nested branch:

```
bzr nested DIR
```

To update the location of a nested branch:

```
bzr nested DIR LOCATION
```

There are multiple reasons for changing the location of a branch:

- to change a read-only location to a read-write location or vice versa
- to change an absolute location to a relative location or vice versa
- to handle genuine moves of the master location
- to refer to a local mirror.

If a nested branch has a read-only location (e.g. a http URL), then local changes cannot be committed to that nested branch. This can be a useful 'firewall' but it's generally more useful long term to ensure read-write locations are used.

There is limited support for specifying locations relative to the location of the containing branch. Relative locations begin with `../` like this:

```
bzr nested lib/osutils ../osutils
```

Relative locations are often more useful than absolute locations because they:

- Make it easier to move a related set of projects.
- Imply the transport used to access nested branches.

For example, if you have a project which is read-only to some (over http say) and read-write to others (over sftp say), then relative locations will ensure both groups fetch nested branches using the best transport for them.

To delete the location of a nested branch:

```
bzr nested --delete DIR
```

The primary reason for using the `--delete` option is to remove a reference to a local mirror. To clear out all values:

```
bzr nested --delete-all
```

To restore default locations after deleting one or more values, simply `pull` the containing branch.

When master locations are moved, it's the responsibility of the project administrators to update the locations stored with the central 'master' branch. Once that is done, they can send email asking users to run `bzr nested --delete-all` followed by `bzr pull`.

Virtual projects

By design, Bazaar is strict about tracking the actual revisions used of nested branches over time. Without this, projects cannot accurately reproduce exactly what was used to make a given build. There are isolated use cases though where is advantageous to say "give me the latest tip of these loosely coupled branches". To do this, create a small 'virtual project' which is just a bunch of *unpegged* nested branches. To mark nested branches as unpegged, use the `--no-pegged` option of the `nested` command like this:

```
bzr nested --no-pegged [DIR]
```

To stop the nested branch tips from floating and to begin recording the tip revisions again, use the `pegged` option:

```
bzr nested --pegged [DIR]
```

After changing whether one or more nested branches are pegged or not, you need to `commit` the branch to record that metadata. (The pegged state is recorded over time.)

For example, you may be managing a company intranet site as a project which is nothing more than a list of unrelated departmental websites bundled together. You can set this up like this:

```
bzr init intranet-site
cd intranet-site
bzr branch --nested bzr://ourserver/websites/research
bzr branch --nested bzr://ourserver/websites/development
bzr branch --nested bzr://ourserver/websites/support
bzr branch --nested bzr://ourserver/websites/hr
bzr nested --no-pegged
bzr commit -m "initial configuration of intranet-site"
```

Publishing the overall site is then as easy as going to the server hosting your intranet and running something like:

```
bzr branch http://mymachine//projects/intranet-site
```

Refreshing the overall site is as easy as:

```
bzr pull
```

Virtual projects are also useful for providing a partial 'view' over a large project containing a large number of subprojects. For example, you may be working on an office suite and have a bunch of developers that only care about the word processor. You can create a virtual project for them like this:

```
bzr init wp-modules
cd wp-modules
bzr branch --nested ../common
bzr branch --nested ../printing
bzr branch --nested ../spellchecker
bzr branch --nested ../wordprocessor
bzr nested --no-pegged
bzr commit -m "initial configuration of wp-modules"
```

Those developers can then get bootstrapped faster and have *just* the subprojects they care about by branching from `wp-modules`.

Nested branch tips & tricks

As explained above, most of Bazaar's commonly used commands recurse downwards into nested branches by default. To prevent this recursion, use the `--no-recurse-nested` option on various commands (including `commit`, `status` and `diff`) that support it.

Thanks to plugins like bzr-svn and bzr-git, Bazaar has strong support for transparently accessing branches managed by foreign VCS tools. This means that Bazaar can support projects where nested branches are hosted in supported foreign systems. For example, to nest a library maintained in Subversion:

```
bzr branch --nested svn://example.com/xmlhelpers src/lib/xmlhelpers
```

If you want revisions to be committed both to a remote location and a local location, make the relevant nested branch a bound branch.

If you need to manage a pile of local enhancements over and above a nested branch maintained upstream, you'll need to use `merge` (and commit), not `pull`, to fetch the upstream changes. Alternatively, consider making the nested branch a loom. See the bzr-loom plugin for details on using a loom.

As you'd expect, a nested branch can be moved or deleted using the normal commands. For example, after splitting out a subproject, you may want to change its location like this:

```
bzr mv src/uiwidgets src/lib/uiwidgets
bzr commit -m "move uiwidgets into src/lib"
```

The `remove` command deletes a nested branch when required like this:

```
bzr remove src/lib/ancientDB
bzr commit -m "delete ancientDB library - no longer used"
```

Things to be aware of

Commands like `commit` and `push` need online access to the locations for nested branches which have updated their tip. In particular, `commit` will update any changed nested branches first and only commit to the containing branch if all nested branch commits succeed. If you are working offline, you may want to ensure your have a local mirror location defined for nested branches you are likely to tweak. Alternatively, the `no-recurse-nested` option to the `commit` command might to useful to commit some changes, leaving the nested branch commits until you are back online.

A given top level branch cannot contain multiple copies of a nested branch. As a consequence, you cannot nest two projects if they both nest the same project somewhere within them. This limitation may be removed in the future. In the meantime, consider restructuring things so that each project is only nested once (and leverage symbolic links as appropriate). In most programming environments, having different parts of the project using different versions of a library is an integration no-no anyhow, so enforcing *one* common revision is the right way to prevent this from happening.

At the moment, nested branches need to be incorporated as a whole. Filtered views can to used to restrict the set of files and directories logically seen. Currently though, filtered views are a lens onto a tree: they do not delete other files and the exposed files/directories must have the same paths as they do in the original branch. In the future, we may add support for nesting and moving selected files from a (read-only) nested branch something like this:

```
bzr nested DIR --file LICENSE --file doc/README::README
bzr commit -m "change which files are nested from project DIR"
```

If you require this feature, please contact us with your needs.