

# Distributed development: toying with Bazaar

Yann Pouillon

CECAM Tutorial — 2008/02/11

*Expected duration: 1 hour*

## Abstract

The purpose of this short hands-on is to discover and play with the simplest among the available Bazaar workflows. Though the project on which you will work has been chosen small for the sake of rapidity and the tasks have been oversimplified, the situation will be very similar to what will happen in a real case.

## 1 First steps

Before actually starting, you're strongly advised to have a look at the Bazaar website, and in particular at the "Docs" section:

```
http://bazaar-vcs.org/
```

You don't need to go into the details. The idea is that you know that this site exists and can help you at any time.

*Exercise 1: (a) have a look at the command-line help; (b) introduce yourself to Bazaar: this first step is essential, as it is how everybody can know who did what.*

## 2 Getting a branch

Let's now imagine that you want to make calculations on quantum dots and have just found on the web a nice little piece of software that perfectly matches your needs:

```
https://code.launchpad.net/~pouillon/etsf-tutorials/  
bzzr-hands-on
```

*Note: actually, this code is a modified version of Alberto Castro's QD code, that you will meet again during the Autotools hands-on.*

The code you're interested in has been published on Launchpad, as part of the "ETSF Tutorials" project. In the world of distributed development with Bazaar, Launchpad can be considered as more or less equivalent to what SourceForge is for Subversion, even though the

underlying concepts remain quite different — for obvious reasons.

Your very first step will be to get the source code. After going to the directory of your choice, you can either follow the instructions from the webpage or directly type:

```
bzr branch lp:~pouillon/etsf-tutorials/bzr-hands-on
```

Now wander a little bit inside the source tree to get familiar with its structure. When inside the "src/" subdirectory, try to build the code. It fails! This is because a nasty guy has introduced and published a very stupid bug somewhere in the source.

*Exercise 2: (a) find all information about the bug and identify the nasty guy using the "bzr log" and "bzr diff" commands and their options; (b) fix the bug and check that the code builds; (c) look at the output of "bzr status"; create a file named ".bzrignore" inside the "bzr-hands-on/" directory containing the following text: "\*.mod"; look at the output of "bzr status" again; (d) commit your changes, writing an adequate changelog message.*

### 3 Branches and repositories

We will now suppose that you like this code and want to contribute to it in two areas: improving documentation and adding a feature. The best way to do it with Bazaar is to create one branch for each topic, and to have them sharing their history in order to facilitate their management. Sharing history is achieved by creating a *repository*.

*Exercise 3: (a) go outside the first branch and create a repository named "qd-contribs"; (b) create two branches from "bzr-hands-on", respectively named "qd-doc" and "qd-feature", inside the repository; (c) look at the files and directories there, and take note of the last modification time of the ".bzr/repository" directory in "qd-contribs".*

Let's now modify a few things inside your new branches:

- add files named "qd-for-dummies.txt" and "qd-devel.txt" containing each a simple line of text into the "qd-doc/doc/" directory;
- add a file named "new\_feature.f90" in "qd-feature/src/" and a file named "new\_feature\_doc.txt" in "qd-feature/doc/", both containing whichever text you want;
- add an empty file in "qd-feature/" by going there and typing "touch ChangeLog.feature".

*Exercise 4: (a) in the "qd-doc" branch, play with the "status", "add" and "commit" commands of bzr in order to commit the "qd-for-dummies.txt" file only: this is called a partial commit; (b) commit the other file as well.*

*Exercise 5: (a) go into the "qd-feature/src/" directory and type "bzr add", then type "bzr status" and understand why there is still one file marked as "unknown"; (b) add the remaining file and commit; (c) look again at the last modification time of "qd-contribs/.bzr/repository".*

## 4 Merging branches

You have now reached the point where you are satisfied with your contributions and would like to put everything together.

*Exercise 6: merge the contents of "qd-doc" into "qd-feature" and commit.*

Everything seems to go very well so far. Then, let's now get into trouble:

- choose two different words;
- add the first word to the "qd-doc/doc/qd-devel.txt" file;
- add the second one to "qd-feature/doc/qd-devel.txt".

*Exercise 7: (a) merge the contents of "qd-doc" into "qd-feature" once again and look at what happens; (b) try to commit; (c) play with the "status", "conflicts" and "resolved" commands of bazaar to solve the conflict, then commit.*

## 5 Publishing changes

During the time your were playing around with Bazaar, the "bazaar-hands-on" branch did not change. As it is the *ancestor* of your branches, you may propagate back your contributions into it by *publishing* them.

*Exercise 8: (a) "push" the "qd-feature" branch into "bazaar-hands-on"; (b) play with the "info" and "log" commands in both branches to understand how publication works; commit a trivial change in "bazaar-hands-on" and "pull" it into "qd-feature".*

## 6 Further reading

We didn't go through all Bazaar commands within this basic tutorial. You may of course learn a lot more by reading the material available from the Bazaar website. There is also a point on which we would like to draw your attention: writing changelogs that can be useful for other developers and maintainers. One very nice way to achieve this is by following the GNU Coding Standards:

[http://www.gnu.org/prep/standards/html\\_node/Change-Logs.html](http://www.gnu.org/prep/standards/html_node/Change-Logs.html)

Happy hacking with Bazaar!