Speaking UNIX: Bazaar

A great place to keep your code

Skill Level: Intermediate

Martin Streicher
Software Developer
Pixel, Byte, and Comma

17 Aug 2010

Software development is characterized by refactorings, quandaries, bugs, epiphanies, and breakthroughs; managing change is essential. Bazaar is a powerful, next-generation source control system that adapts to the dynamics of any development team. This article is an introduction to Bazaar.

Mark Twain once noted, "The time to begin writing an article is when you have finished it to your satisfaction. By that time, you begin to clearly and logically perceive what it is that you really want to say." Twain's observation is right on the mark (no pun intended), and his wisdom applies just as well to software. Invariably, the subtleties, nuances, and scope of the problem at hand become clear only after the application has been written. Between typing the first line of code and packaging a release, there are any number of false starts, dead ends, rewrites, refactorings, quandaries, bugs, and ultimately, epiphanies and breakthroughs.

Frequently used acronyms

- **API**: Application programming interface
- **FTP**: File Transfer Protocol
- **HTTP**: Hypertext Transfer Protocol
- **SFTP**: Secure File Transfer Protocol
- **URL**: Uniform resource locator

Indeed, change is so constant during development that special software must be

used to track modifications. Dubbed *source control* (or *version control*), this software records each and every revision made to the application. Source control lets you compare revisions, switch between one version of a file and another, and gather revisions into a release. And while features, paradigms, and best practices vary from one source control system to another—many proprietary and open source options are available—the intent is universal: Audit who did what and when. Popular commercial source control software includes Perforce and AccuRev; leading open source control software includes Subversion, Arch, Mercurial, Git, and Bazaar (see Resources for links).

Of the latter options, Bazaar is notable, because it's employed to produce the Ubuntu Linux® distribution—an enormous software project with thousands of components. Bazaar is special, too, because it can be adapted to virtually any team dynamic. For example, some development teams prefer centralization, where all changes are collected in and applied to one repository (Subversion is a centralized system). Other teams prefer decentralization, where each developer has an independent repository that can be shared as if it were the principal repository. In this scheme, one collectively owned repository can be designated as a hub for integration, but it differs only by convention (Git is a decentralized system). Another popular work style designates one developer as the "gatekeeper," who has sole access to a canonical repository. Changes are submitted to the gatekeeper who assembles, reviews, approves, and applies modifications to the master repository. Bazaar is flexible enough to accommodate each of these protocols. Like Subversion, Bazaar can sync with a central repository. Like Git, work can proceed locally, detached from any server.

Bazaar also has many other appealing features, such as an API for integration with other software tools, plug-ins to integrate Bazaar with Git and Subversion (among other systems), and a simple numbering scheme for branches that directly reflects the genealogy of each branch. You can also pick up Bazaar in a snap, making it a great choice for versioning your shell "dot" files, documents, and system configuration files. The documentation for Bazaar is outstanding—clear, concise, and practical.

Welcome to the Bazaar

Bazaar is available for all major operating systems. Source code is available, if you prefer to build from scratch, or you can download a suitable binary from the Bazaar home page (see Resources for links). Bazaar is a collection of command-line utilities, but the Bazaar community has also crafted a number of graphical utilities, if you prefer point-and-click interaction. You can accelerate and facilitate adoption with the point-and-click tools and switch to the command-line tools, as needed.

If you're using a UNIX® or Linux system, chances are that your distribution offers a pre-built Bazaar package—typically abbreviated *bzr*; Bazaar-related packages use

the same shorthand for a prefix. For instance, if you use Ubuntu or any Debian-derived Linux version, you can find and install Bazaar software quickly with the Advanced Packaging Tool (APT). Use `apt-cache` to search for *bzr*, as shown in Listing 1.

Listing 1. Find all the Bazaar-related tools in the APT repository

```
$ apt-cache search bzr
bzrtools - Collection of tools for bzr
bzr-builder - construct a bzr branch from a recipe
bzr-cvsps-import - CVS to Bazaar importer
bzr-email - Notification email plugin for Bazaar
bzr-fastimport - Fast-import/fast-export plugin for Bazaar
bzr-git - Bazaar plugin providing Git integration
bzr-gtk - provides graphical interfaces to Bazaar (bzr) version control
bzr-loom - Focused patch plugin support for Bazaar
bzr-pqm - bzr plugin to submit an email to a Patch Queue Manager
bzr-rebase - Rebase plugin for Bazaar
bzr-search - search plugin for Bazaar
bzr-stats - statistics plugin for Bazaar
bzr-svn - Bazaar plugin providing Subversion integration
bzr-upload - Bazaar plugin for uploading to web servers
...
bzr - easy to use distributed version control system

# Install the core bzr package
$ apt-get install bzr
```

The latter shell command downloads and installs the core Bazaar components.

If your operating system does not provide Bazaar, it is easy to build from scratch, especially on UNIX systems. To begin, download the Bazaar source tarball and unpack it into a local scratch directory. Next, ensure that your system has Python version 2.5 or later and the `cElementTree`, `paramiko`, and `Pyrex` modules for Python. Given the source and those modules, type the following to install Bazaar:

```
python setup.py install
```

The installation adds Bazaar to /usr/local/bin. If you want to install the software elsewhere, use the option `--home`, and specify the full path name to the directory, as in `python setup.py install --home $HOME`:

```
$ wget http://launchpad.net/bzr/2.1/2.1.0/+download/bzr-2.1.0.tar.gz
$ tar xzf bzr-2.1.0.tar.gz
$ cd bzr-2.1.0
$ sudo python setup.py install
```

You control Bazaar through a single utility, aptly named `bzr`. For a list of the most commonly used commands, type `bzr help`, as shown in Listing 2. If you want help with a specific command, type `bzr help` *`command`*, where *`command`* is the name of the command, such as `bzr help init`.

Listing 2. Available Bazaar commands

```
$ bzr help
Bazaar -- a free distributed version-control tool
http://bazaar-vcs.org/

Basic commands:
  bzr init           makes this directory a versioned branch
  bzr branch         make a copy of another branch

  bzr add            make files or directories versioned
  bzr ignore         ignore a file or pattern
  bzr mv             move or rename a versioned file

  bzr status         summarize changes in working copy
  bzr diff           show detailed diffs

  bzr merge          pull in changes from another branch
  bzr commit         save some or all changes
  bzr send           send changes via email

  bzr log            show history of changes
  bzr check          validate storage

  bzr help init      more help on e.g. init command
  bzr help commands  list all commands
  bzr help topics    list all help topics
```

A series of Bazaar commands is called a *recipe*, and the online help provides countless recipes to jump start usage and teach best practices. For instance, the material for `bzr help init` provides a recipe to transform any directory into a Bazaar repository, as Listing 3 shows.

Listing 3. Recipe for transforming a directory into a Bazaar repository

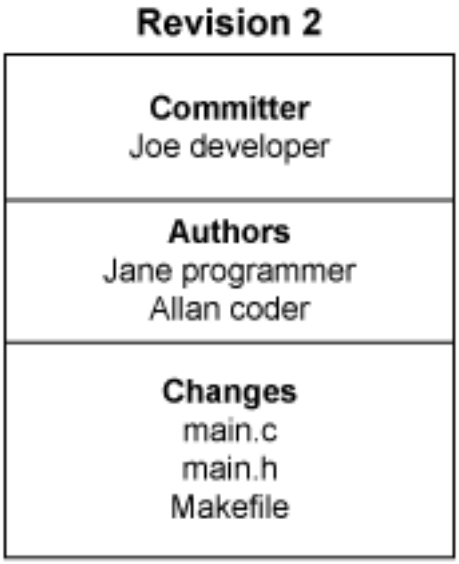
```
$ cd ~/project
$ bzr init
$ bzr add .
$ bzr status
$ bzr commit -m "imported project"
```

This recipe has five steps. To follow along, first change to the project directory you want to keep under source control. Next, run `bzr init` to add a subdirectory named *.bzr* to the current directory to contain the bookkeeping metadata for the repository. The `bzr add .` command recursively adds the contents of "dot" (the current directory) to the manifest of files the repository manages. However, it does not create a snapshot of the current state of the files. `bzr commit` actually records the state of all files in the repository and adds a brief annotation. You can provide the annotation as an argument to `-m` (be sure to use quotation marks to keep the sentence intact) or omit the option and create a note in your favorite text editor. `bzr status` reports what's ready to add to the repository, what's been changed, what's been elided, and more.

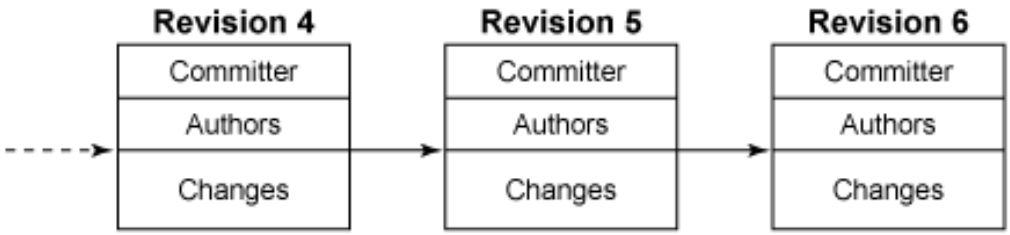
Core concepts

Bazaar shares much in common with other source control systems, although its implementations differ and, pleasingly, leverage facilities you already know how to use (such as Secure Shell (SSH) and URLs). Bazaar manages revisions and can diverge and merge to create and collect branches, respectively. Here are some of the key concepts in Bazaar:

- A *revision* is a collection of changes made to one or more files. You might make a revision to fix a bug or implement a new feature. Each revision has a unique ID (more on IDs momentarily), a sole *committer,* and zero or more *authors.* The committer is the developer who physically recorded the revision, while each author receives an attribution recognizing his or her contribution to the revision. Figure 1 shows a single revision.
  Figure 1. A Bazaar revision

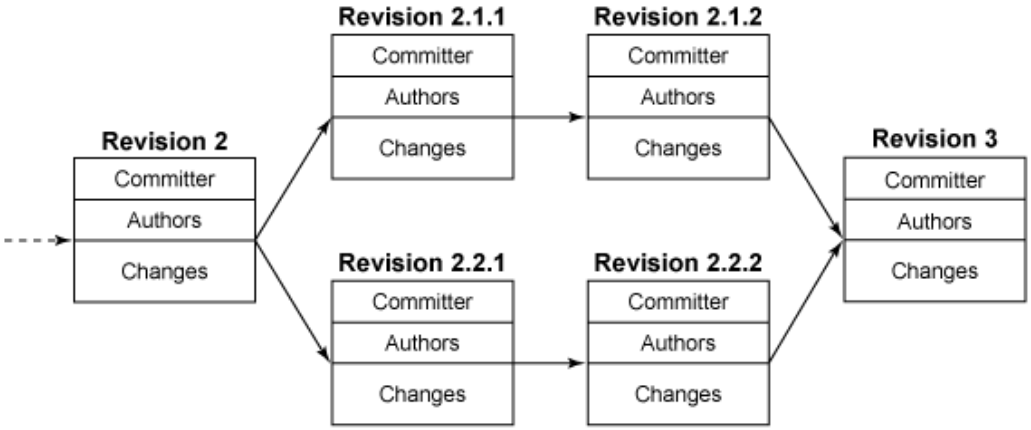


- Typically, each revision has a single predecessor or *parent*. Thus, you can picture development as a chain of revisions, where each revision builds on the changes captured in its parent. This scheme is shown in Figure 2.
  Figure 2. Some development proceeds as a series of revisions.

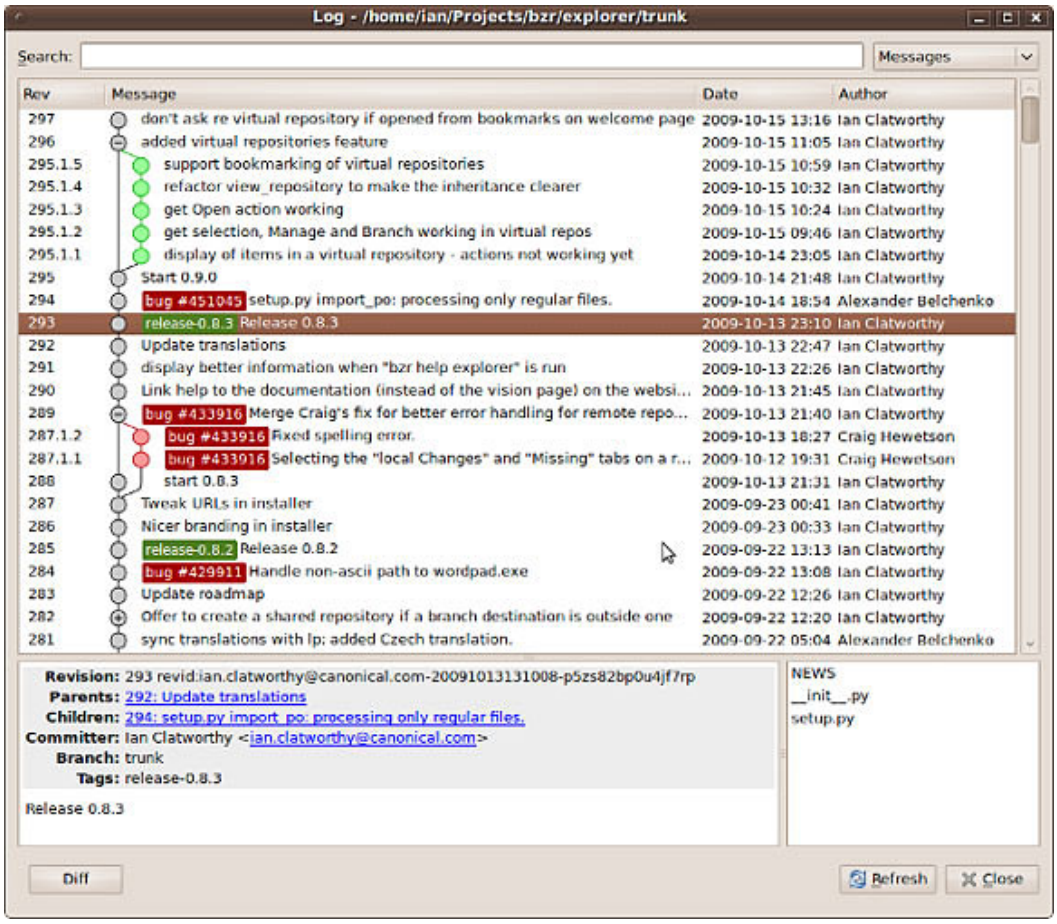


However, development is rarely so orderly, especially when many developers are working in parallel. A revision can have multiple predecessors, too. In this case, a revision represents a merge (see Figure 3 below).

Figure 3. Combining threads of development leads to a merge, a revision with multiple parents



- As mentioned above, a revision ID directly reflects the revision's lineage. Revisions are numbered sequentially starting from 1 and are suffixed each time a chain of development diverges or *branches*. For example, in Figure 3, revision 2 diverges into two branches: *2.1* and *2.2.* The first revision in branch 2.1 is 2.1.1; the second in the same branch is 2.1.2, and so on. When two or more branches merge, the suffixes are dropped and numbering returns to the previous scheme. Revision 3 is the result of merging two branches. Figure 4 shows a graphical tool that pictures Bazaar histories in much the same way as Figure 3. Bazaar's revision numbering scheme is far more meaningful and concise than, say, Git's, which uses long, cryptic MD5 hashes as unique identifiers.
Figure 4. A graphical tool pictures Bazaar histories.



- A *branch* is simply a pointer to a *tip revision*, or a revision that has no successor in the same lineage. In Figure 3, revision 3 represents a branch, because it's the latest revision. In addition, revisions 2.1.2 and 2.2.2 are branches, because both are the ultimate revision in the 2.1 and 2.2 lineages, respectively.

One of the strengths of Bazaar is its reuse of existing, familiar paradigms. A branch is simply a folder. You'll find other examples of common sense features throughout Bazaar. For example, if you use Bugzilla to track bugs, you can annotate a revision as a bug fix with the `--fixes` option.

Performing typical tasks

In general, Bazaar manages entire directories of files. However, you can avoid doing so if you want to omit certain files from version control or operate on individual files or subdirectories. You need not use dot (`.`) to operate on the entire current directory, either initially or during the lifetime of the repository. Many Bazaar commands take a list of file and directory names and operate only on the specified entities.

For example, if you changed files easy.c, medium.c, and hard.c but only want to commit the latter, simply name the file in the commit:

```
$ bzr commit -m "Fix bug no. 99." easy.c
```

Better yet, Bazaar provides a special feature to group and isolate operations to particular files. A *view* is a collection of files and directories. When you enable a view, only the files in the collection are visible. You can create multiple views and switch among them, and you can disable and enable a view to reveal and hide files as necessary. As the Bazaar documentation summarizes, "Defining a view does not delete the other files in the working tree—it merely provides a 'lens' over the working tree." Commands like `bzr commit`, `bzr add`, and `bzr ls` restrict operation to the current view. Other commands that operate on the file system are unaffected by views.

To ignore types of files, such as object files created by compilation or backup files that your text editor generates, you can create an *ignore file* named *.bzrignore* at the top level of your repository. Put this file under version control, too, to help collaborators avoid adding spurious files to the repository. The command `bzr ignore` adds entries to the file for you. The ignore file does not omit files already under version control; it omits only those not yet in the repository.

Oftentimes, a bug fix or enhancement spans multiple files and various sections of code in each file. Unless a developer is very careful to make discrete commits, features can become commingled making it difficult to connect a batch of changes to a bug report or enhancement request. Or, more frequently, a developer may find it necessary to revert a file to a particular state to isolate a bug. In both cases, the developer may want to tease individual revisions from the whole. Bazaar provides *shelves* just for this purpose. You can shelve, or set aside, revisions; make commits (if necessary); and then *unshelve*, or restore, the changes to pick up where you left off previously. Cleverly, Bazaar shelves individual diffs.

Here's one possible scenario. Assume that the file dictionary.txt contains three changes since it was last committed to the repository. You can compare the current state of your work with the repository using `bzr diff`, as shown in Listing 4.

Listing 4. Compare current work against the repository version

```
$ bzr diff
=== modified file 'dictionary.txt'
--- dictionary.txt 2010-04-10 23:57:00 +0000
+++ dictionary.txt 2010-04-10 23:58:40 +0000
@@ -1,2 +1,5 @@
+bonzo
 chico
 groucho
+harpo
+zeppo
```

As the diff shows, the first, fourth, and fifth lines are new. To set aside the changes, run `bzr shelve`. The command interactively prompts you to persist or shelve each diff. If you want to annotate a shelf, provide the `-m` option again, as shown in Listing 5.

Listing 5. Annotate a shelf

```
$ cat dictionary.txt
bonzo
chico
groucho
harpo
zeppo

$ bzr shelve -m 'Save other brothers for later.'
--- dictionary.txt 2010-04-10 23:57:00 +0000
+++ dictionary.txt 2010-04-11 00:15:56 +0000
@@ -1,2 +1,5 @@
+bonzo
 chico
 groucho
+harpo
+zeppo
Shelve? [yNfq?] y
Selected changes:
 M  dictionary.txt
Shelve 1 change(s)? [yNfq?] y
Changes shelved with id "1".
```

If you now examine the file, you should see just two lines. You can also use `bzr shelve --list` to see the list of shelves you have available (see Listing 6).

Listing 6. View lists of shelves

```
$ cat dictionary.txt
chico
groucho

$ bzr shelve --list
  1: Save other brothers for later.
```

The inverse of `shelve` is `unshelve`. This command restores the file to its previous state, as shown in Listing 7.

Listing 7. Restore a file to its previous state

```
$ bzr unshelve
Using changes with id "1".
Message: Save other brothers for later.
 M  dictionary.txt
All changes applied successfully.
Deleted changes with id "1".
```

```
$ cat dictionary.txt
bonzo
chico
groucho
harpo
zeppo
```

Sharing your work with others

Although Bazaar can realize a centralized version control scheme, it is perfectly capable of powering large-scale, distributed software development. Sharing is surprisingly simple and uses any number of existing UNIX sharing technologies.

The simplest technology—and the default unless you specify another choice—is the local file system. You can share your branch with another developer using the handle `file://path/to/your/branch`, where *`path/to/your/branch`* is the absolute path name to your branch, such as /Users/martin/code/emailer. Given the handle, another developer can create a branch based on your work with a single command:

```
$ bzr branch file://Users/martin/code/emailer
```

The `branch` command copies the complete history of this branch so you have everything you need to merge, branch, scan history, and more. Because a branch is just a folder, you need not use `bzr branch`. You can also copy a branch via `cp -pr`, expanding a tarball, or through a remote copy using something like rsync. You can also share a branch via SFTP, HTTP, FTP, and a special protocol built for Bazaar, prefaced with `bzr://`. (For a complete list of supported protocols, type `bzr help urlspec` at the command line.) Once you have a branch, you can update it to match the state of the original using the command `bzr pull`.

Publishing a branch is as simple as sharing the branch's URL. You don't need a special server to publish a Bazaar branch, just make the branch's files and the .bzr directory available via any of the methods already mentioned. You can also push a branch (or the changes for a branch) with a special Bazaar command:

```
$ bzr push sftp://servername.com/path/to/directory
```

Each of the commands that work with branches accept a URL.

Rich offerings from Bazaar

Bazaar has too many features to list here. For a complete introduction and a large

catalog of compatible software, see the Bazaar website. Bazaar also provides a "smart server" that tunnels a special Bazaar protocol via SSH. You or your systems administrator can establish a smart server to host your projects on your own source code server. The administration documentation also explains how to combine Bazaar with plug-ins that amend core features. Moreover, the software provides *hooks* where you can inject your own scripts and code to perform other actions, such as deploying code or running continuous integration tests.

Resources

Learn

- Speaking UNIX: Check out other parts in this series.
- Bazaar online documentation: Learn how to use Bazaar.
- Launchpad: This free source code hosting platform is intended to share open source code. It offers integrated bug tracking, source code reviews, mailing lists to coordinate teams, and more.
- Canonical: Bazaar and Launchpad are sponsored by Canonical, the purveyors of Ubuntu Linux.
- AIX and UNIX developerWorks zone: The AIX and UNIX zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- New to AIX and UNIX? Visit the New to AIX and UNIX page to learn more.
- Technology bookstore: Browse the technology bookstore for books on this and other technical topics.

Get products and technologies

- Bazaar: Download the Bazaar software from the project home page.
- Git: Git is a popular version control system and is used to coordinate development on the Linux kernel, among other substantial projects. Git was created by Linux founder Linus Torvalds.
- Subversion: This longstanding open source version control system is available for nearly every operating system. Its repositories are centralized. A number of graphical Subversion clients are available.
- Mercurial: Mercurial is another recent entrant in open source version control. It runs on all major operating systems.

Discuss

- Follow developerWorks on Twitter.
- Get involved in the My developerWorks community.
- Participate in the AIX and UNIX® forums:
  - AIX Forum
  - AIX Forum for developers
  - Cluster Systems Management

- Performance Tools Forum
- Virtualization Forum
- More AIX and UNIX Forums

About the author

Martin Streicher

Martin Streicher is a freelance Ruby on Rails developer and the former Editor-in-Chief of *Linux Magazine*. Martin holds a Masters of Science degree in computer science from Purdue University and has programmed UNIX-like systems since 1986. He collects art and toys. You can reach Martin at martin.streicher@gmail.com.