Chapter 11

CVS versus BitKeeper—A Comparison

Since the publication of the second edition of this book, a powerful new versioning system has risen called Bitkeeper, or BK/PRO, to dominate at least certain areas of the OpenSource world. BitKeeper was developed by Larry McVoy and is owned and sold by BitMover, Inc. of San Francisco. BK/Pro is a scalable configuration management system, supporting globally distributed development, disconnected operation, compressed repositories, change sets, and repositories as branches.

The distributed development feature allows each developer to get his or her own personal repository, complete with revision history. This tool also handles moving changes between repositories. SSH, RSH, BKD, HTTP, and/or SMTP can all be used as communication transports between repositories. If both repositories are local, the system just uses the file system. For example, the following command would update from a remote system to a local file system using ssh:

```
bk pull bitmover.com:/home/bk/bk-3.0.x
```

Change sets are a formalization of a patch file (i.e., one or more changes to one or more files). Change sets also provide built-in configuration management. The creation of a change set saves the entire state of your repository, both what changed and what didn't, in less than a second.

Today, the Linux kernel itself is managed with BitKeeper and a certain number of other OpenSource projects are switching to this tool, also. It is therefore important to understand how to use it when switching from CVS to BitKeeper.

A Sample BitKeeper Session

A sample session with BitKeeper running on the Linux kernel shows best where the differences are to our beloved CVS. The first step, as with CVS, is obviously to download BitKeeper from **http://www.bitmover.com/cgi-bin/download.cgi/.**

Once your BitKeeper executable is in place, you need to obtain a local clone of the master `BitKeeper` tree by using the following command:

```
bk clone bk://linux.bkbits.net/linux-2.4
```

Beware, however, as `cvs co` and `bk clone` are not completely functionally equivalent. While `cvs co` checks out a working copy of a version of the repository—TOT or a version based on a tag, date, or revision number—`bk clone` makes a copy of the repository itself. Cloning a repository is like doing `cp -r /path/to/cvs/repo/* ~/myrepo/` (not that you'd ever do that!). `bk clone` creates a private *instance* of the repository from (and to) which you can checkout, modify, merge, remove, and check in files.

Later on, when you just want to update your local cloned tree, you can just type the following command:

```
bk pull
```

Here are some other examples of how you can work with files in your repository clone. For each command, a description of the command is provided:

- `bk get` *`filename`*
(Read-only checkout; similar to `cvs export`.)

- `bk edit` *`filename`*
(Read-write checkout; just like `cvs co`.)

- `bk edit` **or** `bk get`
(Checks out all files in the current directory.)

- `bk -r edit` **or** `bk -r get`
(Checks out entire repository.)

- `bk clean`
(Removes all *unmodified* files checked out with `bk edit` or `bk get`. Only operates on files in the current directory.)

A Comparison of CVS and BitKeeper

The remainder of this chapter is devoted to showing you the differences between CVS and BitKeeper. Table 11.1 shows how key features, such as file and directory renaming, GUI tools, Web project tracking, upgrades, and others are implemented in both CVS and BitKeeper.

Table 11.1 Comparing Features in CVS and BitKeeper.

FEATURE	OTHER SCM	BITKEEPER	BENEFIT
Inherently reliable through replication	No	Yes	No downtime. Your developers spend their time developing your product, instead of waiting on a server rebuild.
File/directory renaming	Rarely	Yes	Increased productivity through well organized source base.
BK/ProMerge™	No	Yes	Accurately reduces the number of merge conflicts and eases resolution of remaining conflicts.
True distributed system	No	Yes	100% productivity at geographically distributed sites at all times, with no loss of functionality or performance. Any user may modify any file on any branch at any time, without restriction.
Powerful GUI tools	No	Yes	Dramatically simplifies debugging, easier merges, improves check in comments.
All changes are reproducible snapshots	No	Yes	Easily remove bad changes, aids in debugging, and aids in release management.
Web project tracking	Maybe	Yes	Allows management to track projects and estimate release dates.
Optimal performance for users, local or remote	No	Yes	Database replication means all all developers, local or remote, get optimal performance. BitKeeper works well even over low bandwidth, high latency links such as modem or satellite links.
Disconnected (laptop)	No	Yes	Productivity while traveling, at home, at remote offices with partial/slow network connectivity.
Peer-to-peer architecture	No	Yes	Work may flow in any direction, including "sideways" between two developers without involving a "master" copy.

(continued)

Table 11.1 Comparing Features in CVS and BitKeeper (continued).

FEATURE	OTHER SCM	BITKEEPER	BENEFIT
Painless upgrades	No	Yes	Upgrading server does not affect developers.
Cross platform GUI	Rarely	Yes	Increased productivity, no retraining.
Scripting	Maybe	Yes	Easily customizable to your environment.
Customizable reports	Rarely	Yes	Accountability and status to/for managers.
Automatic integrity checks	No	Yes	Catches hardware/software problems promptly, while replicas are still available.
Integrated bug tracking**	Maybe	Yes	Link bugs to changes and vice versa.
Active roadmap	Maybe	Yes	BitKeeper is actively developed by a world class development team. Follow on products for bug tracking, sales tracking, project management, and project hosting are all actively being developed.

Comparing Commands and Syntax

As you are moving from CVS to BitKeeper there are some differences in commands and command syntax that you should be aware of. I've included the following command comparison chart to help you work with both tools.

Style Conventions:

- Commands are `styled in teletype`.
- Additional information about a command is (enclosed in parentheses).
- User-defined command arguments are *styled in italic*.

cvs co *directory or module* bk get or bk edit *filename*	**bk clone bk://*server:port/parent_repo parent_name***

Details:

BitKeeper performs filename expansion in the following order:

1. `bk command dir`

   (When directory is specified and if dir/ exists, then implied file list is dir/SCCS/s.*)

2. `bk command <NULL>`

   (When no directory or files are specified and if SCCS/ exists, then implied file list is SCCS/s.*)

3. `bk command [file1 file2 file3]`

   (When one or more files is specified, then each file name is converted to SCCS/s.file1 SCCS/s.file2, etc.)

Some commands accept an option `-r` flag. Here's the implied behavior:

`Bk command`	=>	`bk command SCCS/s.*`
`Bk -r command`	=>	`cd 'bk root'; bk sfiles | bk command`
`Bk -r. command`	=>	`bk sfiles | bk command`
`Bk command dir`	=>	`bk command dir`

cd ~/mytree; cvs update	**cd ~/myclone; bk pull bk://*server:port/reponame***

Your private repository is a "child" of the "parent" repository that you cloned.

Use `bk pull` to update your repository with changes committed to the parent since the time that you either created your clone or last pulled.If merge conflicts occur during the pull,

Bitkeeper will report them. Use `bk resolve` to resolve the conflicts.

cvs add [-kb]	**bk new [-b]**

bk new adds and checks in one or more files to your repository, placing it under revision control. Like cvs add, bk new is not recursive.

- `bk new filename`

  (add a single file)

- `bk sfiles -x | bk new -`

  (add multiple files [bk sfiles -x finds all files not under revision control])

- `bk sfiles -x *.[ch] | bk new -`

  (add multiple files in current directory based on a pattern)

- `bk new -b`

  (add a binary file, similar to cvs add -kb. Turns off keyword expansion)

To add many files in multiple directories, try this:

- `cd ~/myrepo`
- `cp -r ~/to_add/* .`
- `bk sfiles -x . > /tmp/LIST`

- `vi /tmp/LIST`

  (remove whatever shouldn't be added)

- `bk new - < /tmp/LIST`

Details:

- In CVS, to add new files and directories, you first `cvs add` the directory, then add each file, then `cvs commit` the directory. In BitKeeper, `bk new` *`filename`* adds and checks in the directory and file in a single step.

cvs commit -m "checkin comment"	bk ci -y"checkin comment" bk commit -y"changeset comment" bk push

No space is allowed between the -y and the quoted comment.

Note: *You are strongly encouraged to use bk citool for all of your check-ins and commits. It's cleaner, easier, and fosters better comments!*

- `bk ci -l -y"checkin comment"` *`filename`*

  (checks in changes and, afterward, does a `bk edit` *`filename`*)

- `bk sfiles -U -c | bk ci -y"your checkin comments" -`

  (check in lots of files from the command line)

Checkins and changesets:

- Use `bk ci` to check in a change.
- Use `bk commit` to define and check in a changeset.
- Use `bk pending` to get a list of checkins that are not grouped in a changeset.
- It bears repeating: `bk commit` is for defining a changeset, **NOT** for checking in individual changes.
- It's okay to use `bk ci` to check in changes and, later, use the graphical bk citool to define a changeset.
- Use `bk push` to propogate changeset to the parent repository.

Details:

What is the general "best-practice" guideline for a defining changeset? That is, when should checkins be grouped into a changeset?

The simple answer is, whenever you need to exchange data with another repository you can update your repository with `bk pull` or promote your changes to your repository's parent with `bk push`. Stated simply, you *cannot* pull or push without first grouping your checked-in changes into a changeset.

cvs remove *filename or directory*	bk rm *or* bk rmdir

The files and directories are moved to BitKeeper's version of the CVS Attic, ~/myrepo/BitKeeper/deleted.

n/a	bk mv

`bk mv` has no corollary in CVS. `bk mv` renames the checked-out file (if any) as well as the revision control file in the SCCS directory. Edited files are also renamed and then re-edited (i.e., checked out), preserving any changes that are not yet checked in. `bk mv` operations appear as a change when you commit the next changeset.

Moves propagate like content changes, that is, they are applied in the next pull or clone. Use `bk mvdir`, not `bk mv`, to rename directories.

cvs diff	bk diffs *or* bk difftool

View differences between your checked-out file and the latest or earlier committed version in your repository.

Just like CVS, `bk diffs` can be used with a revision number, as in `bk diffs -r1.2` *filename.ext*.

Details: How can I get a diff of changes between my checked-out, locally modified files and what's in the *parent* repository? The short answer is, you can't—not without pulling an update from the parent. That said, if the parent has changesets that you do not have and if both the parent and child repository are on the same file system, you could get a meaningful diff with `bk treediff`.

cvs history	bk cmdlog [-a]

Displays the history of repository-level commands run in the repository for the current directory. Repository level commands are `clone`, `commit`, `export`, `pull`, and `push`.

cvs import	bk import *or* "bk extras | bk new -"

See **cvs add / bk new** above.

cvs log	bk prs *or* bk changes

`bk prs` shows the revision summary for a single file or directory of files. It is not recursive.

`bk` changes shows the changeset history for the entire repository. Again, this command operates on the entire repository, even if you `cd` deep into your tree and specify a single file.

cvs release *filename or directory*	bk ci -y"comment" *filename or directory* bk clean

`bk edit` checks out and locks a file for modification. To release the lock, check in your changes and run `bk clean` to remove the file. If don't want to keep your modifications, use `bk unedit` *`filename`* to unlock the file (use with caution!).

Details:

- A checkout "lock" blocks you from running `bk edit` on the same file more than once, thereby avoiding overwriting uncommitted changes with a new checkout.

cvs status	bk status [-v]

Analyzes the status of your checked-out tree. Verbose mode, `-v`, lists the parent repository, users, files not under revision control (extras), files modified and not checked in, and files with checked-in-but-not-committed deltas.

cvs status | grep "nothing known about"	bk extras or bk status –v

Finds files that are not under revision control.