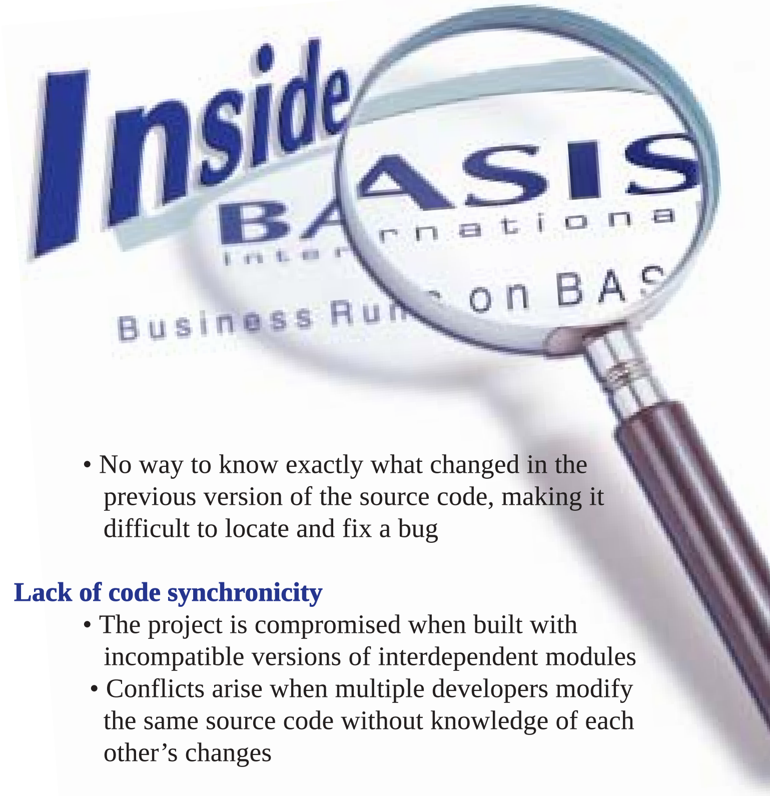


Software Product Version Control

By Earl Box

A manager has a large software development project; the development staff is onboard, planning is complete, and now it is time to turn the programmers loose to put the plan into action. Would it be helpful to have a software solution to help store and maintain the code that these programmers write, rewrite, and debug? What if it could also track all changes, maintain the code in a single repository, and provide the ability to build the product from a single location? Source code version control software does all this, and more.

Many organizations structure their projects so that individual programmers have responsibility for particular segments of the product. These developers often maintain their individual source code and supporting software on their own computer system. When it comes time to produce a releasable version of the product, each programmer must put his or her segment of the product into a centralized directory on a common machine for final compilation. This scenario has many pitfalls:

Lost source code due to

- Aggressive code refactoring resulting in overly complex and buggy code
- Accidental file and directory deletions
- Hardware failure
- Stolen computers
- Recent backups unavailable

Halted development when the programmer is absent or unavailable

- The only copy of the source code is on the programmer's laptop, which is with the absent programmer
- The programmer's machine is in the office, but the login and password is unknown preventing access to source code
- Several versions of the source code exist in multiple directories - which is the most current version?
- Even if someone finds the source, how does the absent developer normally build it – from the IDE, through scripted routines, or another specialized process?

Lack of history

- No way to identify the reason for the code change. Did the change fix a specific bug, add new features, or was it a major rewrite?
- No way to know exactly what changed in the previous version of the source code, making it difficult to locate and fix a bug

Lack of code synchronicity

- The project is compromised when built with incompatible versions of interdependent modules
- Conflicts arise when multiple developers modify the same source code without knowledge of each other's changes

Version control software effectively addresses all of these potential pitfalls and plays an important role in any comprehensive software development cycle. Perusing the internet reveals thousands of open-source projects that are stored in Concurrent Versions Control (CVS) archives. The CVS system manages the source code changes, even with thousands of programmers modifying the code on the same project. Quality source code control allows multiple developers around the world to work simultaneously on a common software project and is the method with which the open source community develops such popular software packages as Linux, FreeBSD, and Apache.

Put simply, version control software keeps a history of the changes made to a set of files. For developers, that means being able to keep track of all the changes made to a program during its development cycle. There are many version control systems to choose from such as SoftLanding's PVCS Version Manager, Microsoft's Visual SourceSafe, and CVS, the system BASIS uses. The brand of version control system chosen depends on the development environment, size of the project, and feature set. While version control systems all provide the fundamental ability to track changes, they can differ in other areas – such as the method used to access files. Exclusive locks and shared access are the two most common file access methods.

Exclusive Locks: A checked out file is available only to the person who checked it out. In other words, when a programmer checks out a file, he or she is the only one who can modify the file in the same way books in a library are available only to the person who checks them out. Once the programmer checks the files back in, other developers can access the files. This access method, however, has its drawbacks. For example, if a developer forgets to check in file

continued...

changes over a weekend, these files are unavailable to other developers who may need to make additional modifications. Fortunately, there are administrative tools that overcome these situations but it can still be a problem from time to time.

Shared Access: Multiple developers have simultaneous modification access to the same file(s). When the developer checks in their working copy, CVS merges the updates from their copy with what is in the archive. Most of the time developers modify separate sections of the code so there are no conflicts. However, during the rare times when two developers change the same lines of code, CVS handles this cautiously. It puts the modifications from the archive and your modification to the same lines into your local copy of the file, with annotations that identify the conflicts. CVS merely identifies the conflict, relying on the developer to resolve the issue.

At BASIS, the central software archive resides on a UNIX machine and the developers access it either from other UNIX, Linux, or Windows machines using the built-in client/server access methods. CVS supports various platforms for both the clients and servers.

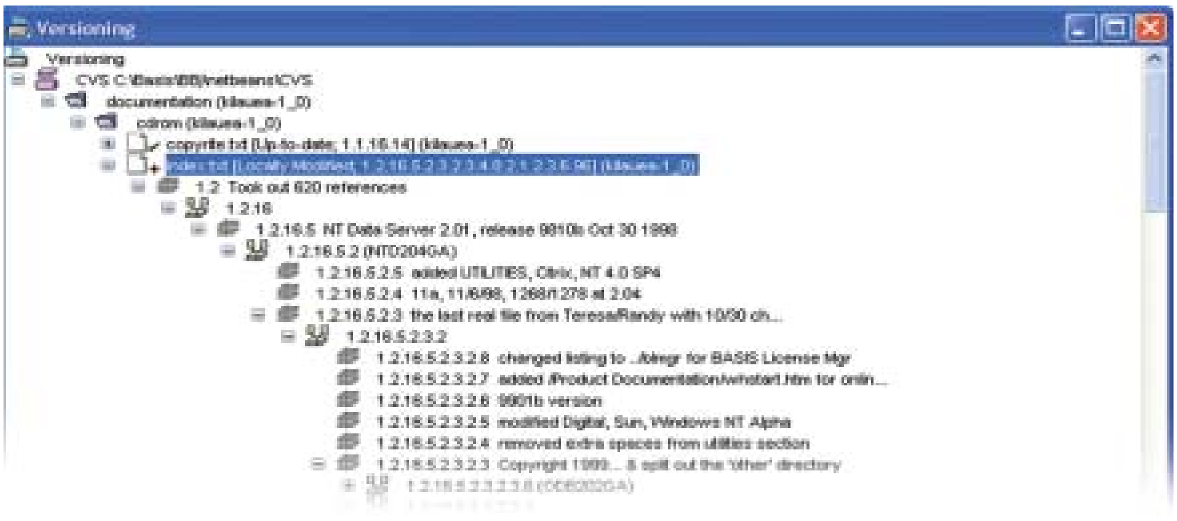
Some of the more powerful concepts and functions that CVS offers include:

Checkout – the process of checking out a copy of one or more files from the archives to a local computer. Developers can check out a single file, a module consisting of several files, or the entire archive at a designated version. The ability to check out different versions allows a developer to build the product from various stages in its life, regardless of how many files other developers added, deleted, or changed in the interim.

Branching – the ability to freeze source code at a specific point in the archive process. Branching allows part of the development team to continue developing future changes/enhancements in the main line of the archive while another part of the team makes changes that only apply to the branch. The branch keeps the two development projects separate, however, it is possible to merge a branch back into the main line later, if desired. Because of this, branching makes it possible to setup a "skunkworks" project development scenario and either discard it or merge the changes into the main project.

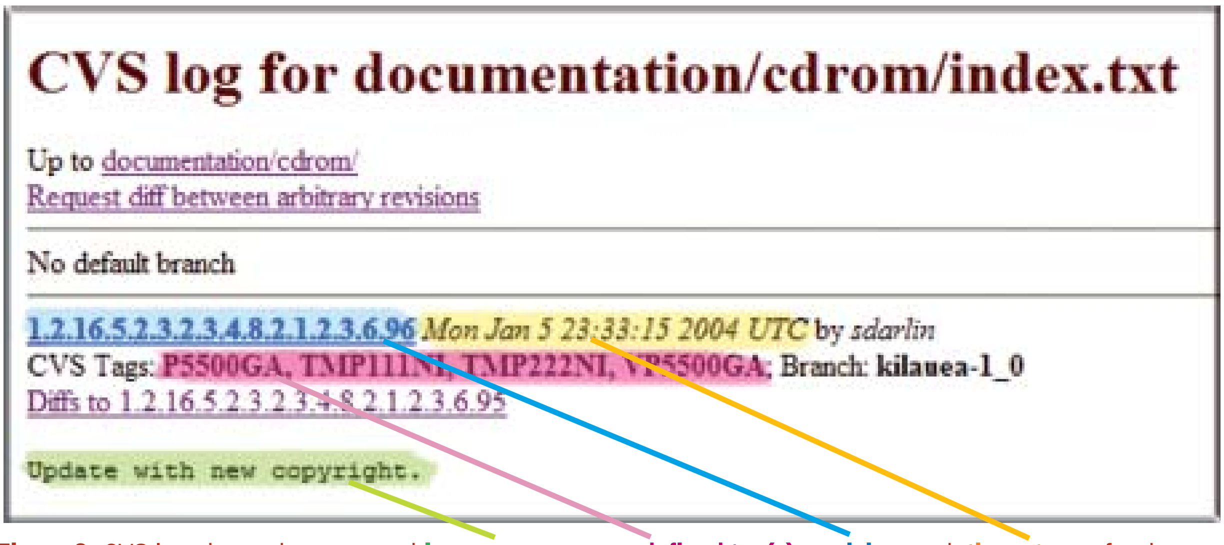
Branching also provides reproducibility, allowing developers to "go back in time" and check out source code as it existed at any point. This makes it possible to recreate a previous version of the product, regardless of the number of subsequent changes, additions, or deletions to the project. The BASIS IDE™ creates a graphical representation of the history of all changes to a file and its branches, as displayed in **Figure 1**.

continued...



```
Versioning
Versioning
  CVS C:\Basis\BB\netbeans\CVS
    documentation (kilauea-1_0)
      cdrom (kilauea-1_0)
        copyrite.txt [Up-to-date; 1.1.16.14] (kilauea-1_0)
        index.txt [Locally Modified; 1.2.16.5.2.3.2.3.4.0.2.1.2.3.6.90] (kilauea-1_0)
          1.2  Took out 620 references
            1.2.16
              1.2.16.5  NT Data Server 2.01 , release 9810b Oct 30 1998
                1.2.16.5.2 (NTD204GA)
                  1.2.16.5.2.5  added UTILITIES, Citrix, NT 4.0 SP4
                  1.2.16.5.2.4  11a, 11/6/98, 1268/1278 at 2.04
                  1.2.16.5.2.3  the last real file from Teresa/Randy with 10/30 ch...
                    1.2.16.5.2.3.2
                      1.2.16.5.2.3.2.8  changed listing to ../blmgr for BASIS License Mgr
                      1.2.16.5.2.3.2.7  added /Product Documentation/whstart.htm for onlin...
                      1.2.16.5.2.3.2.6  9901b version
                      1.2.16.5.2.3.2.5  modified Digital, Sun, Windows NT Alpha
                      1.2.16.5.2.3.2.4  removed extra spaces from utilities section
                      1.2.16.5.2.3.2.3  Copyright 1999... & split out the 'other' directory
                        1.2.16.5.2.3.2.3.6 (ODB202GA)
```

Figure 1. Graphical display from the BASIS IDE of the lifetime versioning of a file



CVS log for documentation/cdrom/index.txt

Up to documentation/cdrom/
Request diff between arbitrary revisions

No default branch

1.2.16.5.2.3.2.3.4.8.2.1.2.3.6.96 Mon Jan 5 23:33:15 2004 UTC by sdarlin
CVS Tags: P5500GA, TMP111NI, TMP222NI, VP5500GA; Branch: kilauea-1_0
Diffs to 1.2.16.5.2.3.2.3.4.8.2.1.2.3.6.95

```
Update with new copyright.
```

Figure 2. CVS log shows the generated **log message**, **user-defined tag(s)**, **revision**, and **time stamp**, for that version.

Tag – the user-defined name assigned to a specific revision of a file(s). Tagging may more intuitively identify a particular version of a file. A developer will more readily remember the tag than an obscure date or the lengthy revision as shown in **Figure 2**.

Log – the overview of the changes (see **Figure 2**) made to a particular file without having to peruse the code. When checking in changes to a file, developers provide a description of the work done. These descriptions give an overview of the changes, or perhaps even mention which bugs or defect numbers the code changes address. Viewing the log of changes makes it easy to see what modifications occurred in the file and why, without having to look at the individual lines of the source code.

Diff – the comparison between two different revisions of a program. This allows the developer to determine when, where, and who corrected a problem, added a new feature, or introduced a particular bug into the source code.

Since a complete archive of a project contains multiple versions of many files, one might conclude that CVS stores a full copy of each file and requires a significant amount of hard disk space. Fortunately, this is not the case. CVS stores the initial ASCII file in its entirety and then, in subsequent revisions, only tracks the changes. Storing these incremental changes dramatically reduces the storage requirements for the archive.

It is difficult to complain about the price of CVS because it is free! The BASIS IDE is also free and includes an integrated CVS client, simplifying the developer's ability to use CVS for all BASIS products. Whether developing a project in PRO/5®, Visual PRO/5®, or BBj®, CVS gives the programmer the ability to maintain safe and accurate version control from within the BASIS IDE. This is just one more reason why the BASIS IDE is the tool of choice for BB^{X} development.

continued...

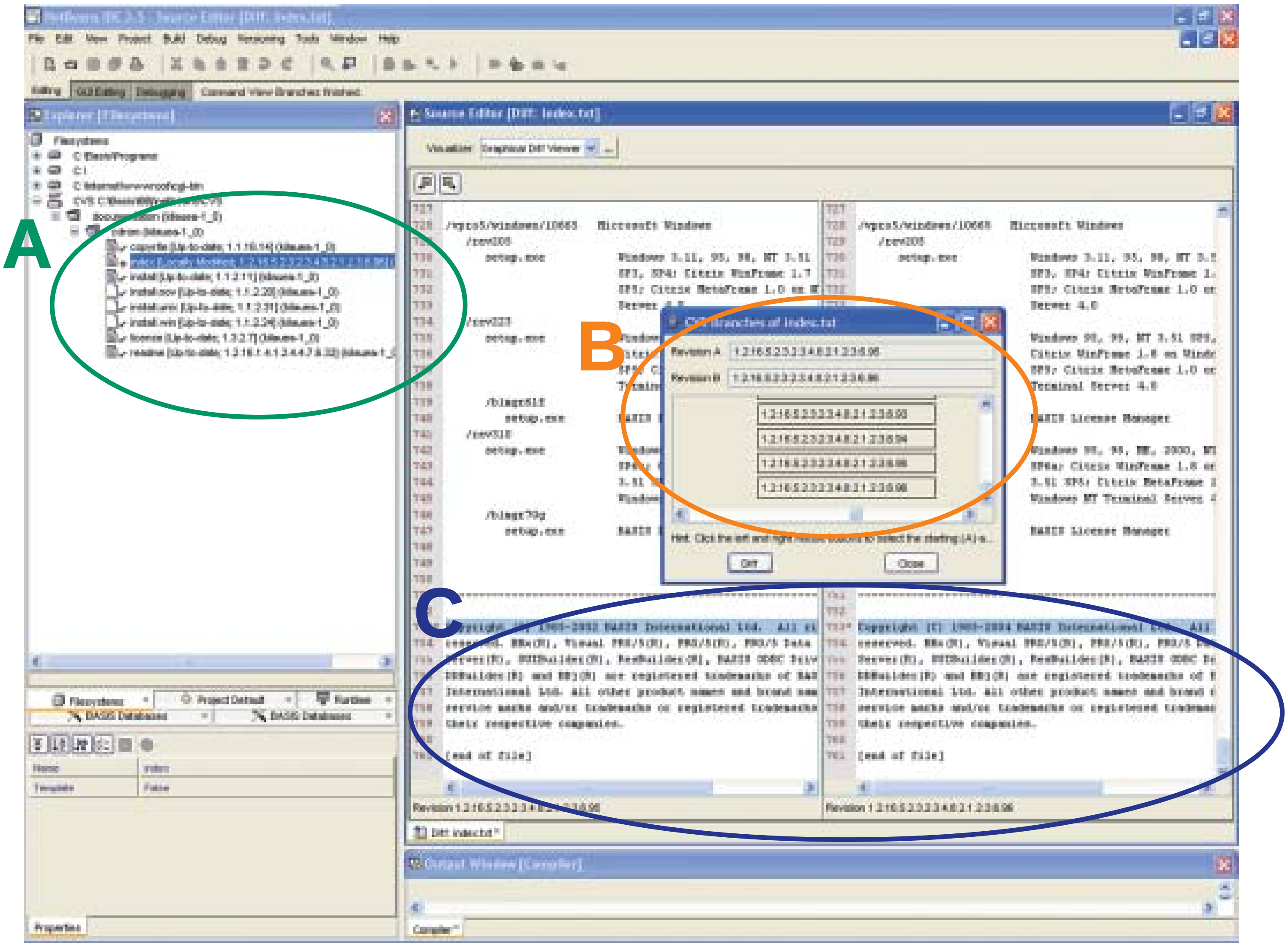


Figure 3. BASIS IDE showing CVS client features

Figure 3 shows the BASIS IDE and these integrated CVS client features:

A. Mounted CVS archive (displays file status modified (+) or updated (√), revision number, and branch),

B. File branch structure (with “diff” dialog to easily select two versions of a file to compare), and

C. The actual “diff” (highlights the changed text in the two versions).

Considered a safety net, a version control system provides recovery opportunities and peace of mind throughout the development cycle. Confidence and security may be the greatest benefits a version control system delivers.

For more information, a free download, or CVS-related resources, go to ***www.cvshome.org***.