Editor: Diomidis Spinellis ■ Athens University of Economics and Business ■ dds@aueb.gr

Version Control Systems

Diomidis Spinellis

A source code control system [is] a giant UNDO key—a project-wide time machine.
— Andy Hunt and Dave Thomas

Sane programmers don't write production code without the help of an editor and an interpreter or a compiler, yet I've seen many software projects limping along without using a version control system. We can explain this contrast if we think in terms of the increased start-up costs and delayed gratification associated with adopting a VCS. We humans typically discount the future, and therefore implementing version control in a project appears to be a fight against human nature. It's true that you can't beat the productivity boost that compilers and editors provide, but four decades after punched-card programming in assembly language has gone out of fashion, we must now look elsewhere for our next efficiency gains. And if you or your project isn't using a VCS, adopting one might well be the single most important tooling improvement you can undertake.

Procurement and installation

Acquiring a VCS need not be expensive; depending on the operating system you're using, you might in fact find out that one is already installed and ready to run—probably CVS (Concurrent Versions System) or RCS (Revision Control System). If not, you have the luxury of a wide choice. If you're on a shoestring budget, you can safely pick a free open source system: multimillion-line projects have relied on such systems for more than a decade. If you can shell out some cash, you'll find that several commercial systems offer additional features and a more polished interface. Installing a VCS typically also involves setting up a *repository*, the location where the definitive version of your source code and its changes will reside. Be sure to include the repository in your scheduled backups.

Life with a VCS

Normal software development with a VCS is only marginally more complicated than without it. Initially, you start out a new project or *import* your existing project into the VCS. From then on, to work, you *check out* a version of the project into your private working directory. Every time you're happy with a change you've made—like a bug fix or the addition of a new feature—you *commit* your change to the repository, along with an explanatory message. (See this issue's Glossary for more VCS terminology.) Also, whenever you feel in the mood for some excitement, you *synchronize*, or *update*, your private version of the software with the changes your colleagues have committed. This action will provide you with endless hours of fun as you battle against your colleagues' mistakes but also ensures that you're all working on roughly the same source code base. Finally, when you roll out a release, you *tag*, or *label*, all files with the release's name. And that's basically it.

The goodies

Having convinced you that adopting a VCS isn't a Herculean task, let's briefly review some

0740-7459/05/$20.00 © 2005 IEEE

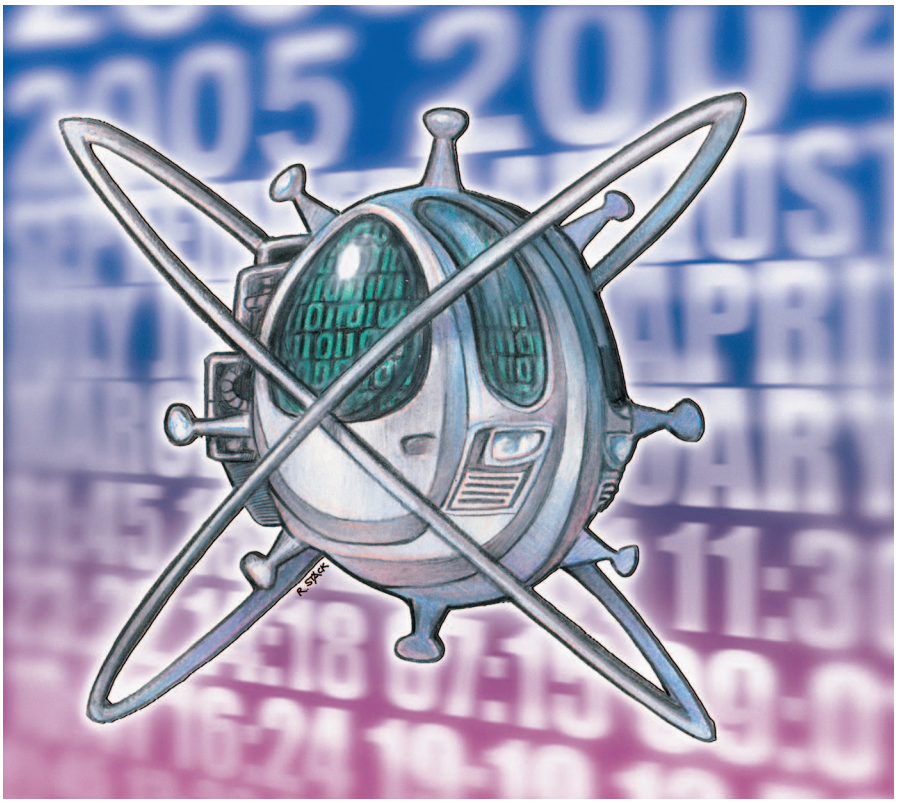
of the benefits you'll reap. First of all, if you're working in a team, you'll stop stepping on each other's toes by writing over other people's code. If both you and Mary change the same file, the system will either unobtrusively merge your changes or warn you that these are conflicting.

In addition, every time you commit a change, you create a new version of the corresponding files. With the version information that the VCS stores, you can access each file's history of changes, see the differences between versions of the same file, and see who changed which lines when. Now that you can always go back to a specific version of the file, you don't need to comment out code blocks "in case they're needed in the future": your older version of the code is safely stored in the VCS repository. And, in many VCS implementations, you can get an annotated listing of the file indicating the developer and date corresponding to each line's most recent change. The repository also acts as the source of truth regarding the files stored in it. Source code distribution simply involves obtaining or updating a private workspace from the VCS repository. Once you label a project's files for a given release, you can use the release's name to obtain again an exact copy of that historic file set.

Furthermore, you can split development into different *branches,* each branch for example tracking the fixes associated with a given software release. You can then easily obtain the file versions associated with a branch and apply the same fix to multiple branches. Finally, with all the project's history neatly stored in the repository, you can mine the VCS data to see how you're doing: How many lines were changed for version 3.1? Which are the most and least productive days of the week? Which developers work on the same files?

Best practices

Even if you've already been using a VCS for some time, you might be able to squeeze more juice out of it. Here are some ideas:



- *Put everything under version control.* Version control isn't only for source code; use it for your build scripts, help files, design notes, documentation, translated messages, GUI elements, and so on—everything that makes up your project.
- *Use a VCS on your personal projects.* You don't have to work on a team to adopt a VCS. Consider using one for your personal files, such as your hobby projects, your Web page, or your phone book. Some developers even use a VCS to synchronize their home directories among different hosts.
- *Think carefully about file naming and organization.* Some VCSs get confused when a file name changes, offering you the unattractive choice of losing either the file's revision history or the ability to retrieve older versions of the software with the correct file name. So, it makes sense to adopt from the start file names and a directory organization that will remain relatively stable throughout the project's life.
- *Perform a separate commit for every change.* Don't lump multiple changes into a single commit. Separating changes lets you see precisely which lines the change affected and apply the change selectively to other branches. This rule is especially important if a change involves global stylistic changes, which will affect thousands of code lines.
- *Label all releases.* Whenever you release the software (even to the testing group next door), label it. This provides everyone with a concrete name to associate with bug reports and their fixes.
- *Establish and follow policies and procedures.* VCS actions can affect all developers. So, you'll benefit from clear policies covering developer etiquette or the content of commit messages and from procedures covering heavy operations such as branching and releases.

$Id: tot-5.rtf 1.4 2005/07/05 15:49:38 dds Exp dds $

Diomidis Spinellis is an associate professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Reading: The Open Source Perspective* (Addison-Wesley, 2003). Contact him at dds@aueb.gr.