



# Git

Diomidis Spinellis

**EVEN IN LIGHT** of our field's dizzying rate of progress, I wouldn't have expected to revisit the subject of version control just seven years after I first wrote about it in this column ("Version Control Systems," *IEEE Software*, vol. 22, no. 5, 2005, pp. 108–109). Yet, here we are. The new kid on the block is git, a distributed revision control system available on all mainstream development platforms through a free software license.

Git, the brainchild of Linus Torvalds, began its life in 2005 as a revision management system used for coordinating the Linux kernel's development. Over the years, its functionality, portability, efficiency, and third-party adoption have evolved by leaps and bounds to make it its category's leader. (Two other systems with similar characteristics are Mercurial and Bazaar.)

## Revisions, Not Versions

Traditional version control systems derive their requirements from software

configuration management practices. The focus of these practices is to identify, control, and disseminate the software's configuration and changes. A system like CVS or Subversion that can retrieve the files corresponding to a specific software version, list the changes that led to it, and keep developers from trampling on each others' feet satisfies these requirements and offers an advantage over exchanging files through a shared folder or email. However, configuration management primarily prevents bad things from happening during software development; in other words, it provides (valuable) control but few tools that genuinely aid a developer's everyday life.

Developers don't really care whether they work on version 8.2.72.6 of branch RELENG\_8\_2, but they care deeply about software revisions: changes they made to fix a specific bug, infrastructure changes that were needed to support that fix, another set of changes that didn't work out, and some work in progress that was interrupted to work on that urgent bug fix. Fittingly for a tool written by a programmer to scratch his own itch, git supports these needs with gusto. It gives developers a complete copy of the software repository, allowing them to create their own private branches corresponding to their individual needs. Each branch can correspond to a distinct task, like the de-

velopment of a new feature or a bug fix. Developers can quickly create and delete branches, switch from one working branch to another, make small incomplete incremental commits, cherry-pick commits from other branches or commits, or even stash away some changes to revisit them later. When a feature is mature for wider distribution, developers can package their changes as a complete well-integrated changeset that others can merge into their work.

An important difference of git over its older ancestors is that it elevates the software's revisions to first-class citizens. By managing revisions, git allows a developer to select precisely which ones will comprise an integrated change, down to partial changes within a single file. More importantly, git keeps as a graph a complete history of what changes have been merged into which branches, thus allowing developers to think in terms of revisions they've integrated rather than low-level file differences between diverging branch snapshots. This switch to a higher level of abstraction is no less dramatic than the one from assembly language, which dealt with CPU registers and memory addresses, to high-level programming languages, which provide entities like objects, containers, and threads. As one would expect, a higher level of abstraction provides opportunities for changing the way we think and work.

Post your comments online  
by visiting the column's blog:

[www.spinellis.gr/tools](http://www.spinellis.gr/tools)

## Decentralized Revision Control

By managing revisions, git makes it natural and easy to push a revision to a remote repository (remember, each developer has a separate complete repository copy) or to pull some revisions from a remote repository to the local one. This in turn allows developers and their managers to build a variety of interesting workflows, most of which are impossible to run on a traditional centralized version control system. For instance, an integration manager can selectively pull changes from developers' public repositories and integrate them into a master repository that contains the project's definitive picture. If the workload on the integration manager becomes excessive, a team of "lieutenants" can take over the integration of specific project parts. The lieutenants integrate the developer changes in their public repositories, and a higher-level manager can then take those larger change sets and integrate them into the master repository. (This is the Linux kernel development model.) Or two developers can coordinate and share their work in a peer-to-peer fashion by pulling from each other's repository.

## The Importance of Being Local

Unfortunately for this column's focus, there's more to git than its superb management of revisions and decentralized repositories. First, by keeping locally a complete version of a repository, git allows you to work and commit individual changes without requiring Internet connectivity. This local staging area also makes it possible for you to edit, reorder, and squash together your past commits (*rebase*, in git's parlance) in order to present a coherent story to the outside world. When you're back online, you can push your changes to a remote repository. The project's entire past history is also always available to you. Want to see who fixed a specific bug while traveling at 30,000 feet? Go

through the project's commit history; it's there. Want to examine how the bug was fixed? The corresponding changes are likewise one command away. Furthermore, the local repository (and, no doubt, some highly skilled programming) makes all operations blindingly fast. This is a blessing for your personal productivity, but it's also an enabler for

offer similar functions. GitHub simplifies many repository management tasks through a Web-based user interface. In addition, it promotes cooperation in open source projects, which are hosted for free, by making it easy for developers to clone existing projects and submit their contributions as a pull request. If you decide to pay to

Git allows developers and their managers to build a variety of interesting workflows.

performing more complex operations. For instance, building on the rapid repository access, git's *bisect* command allows you to perform a binary search between two points in time to find the commit that broke your software. Finally, local repositories make it trivial to put even the smallest personal project under version control. Just enter "git init" at the directory in which your project resides, and you're ready to go. When you later decide to share the project with others, you can easily associate it with a public remote repository and push there all your changes. This (plus git's ability to import history from other version control systems) has allowed me to share work that precedes git's inception.

## Enter GitHub

If the idea of setting up a public repository, maintaining its servers and connectivity, keeping it secure and up to date, setting up user accounts, and supporting your users isn't appealing, you can delegate all such tasks to a third-party provider. GitHub is the best known, but at least eight others

host a proprietary project on GitHub, then you'll value the ability to set up teams with varying access rights across the project's repositories. GitHub also provides an issue-tracking system, a file download area, and Gollum, a git-based wiki. Through Gollum, you can edit a page on the Web and record the change as a git commit, but you can also perform manual or automated changes on the files of a local wiki clone and then push them onto an upstream repository. This gives you wiki-style effortless collaboration with git's workflow sophistication; what more could one want? 

**DIOMIDIS SPINELLIS** is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of the books *Code Reading* and *Code Quality: The Open Source Perspective* (Addison-Wesley, 2003, 2006). Contact him at [dds@aueb.gr](mailto:dds@aueb.gr).



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.