A Comparison of Centralized and Decentralized Version Control Systems

Brian de Alwis (University of Saskatchewan)  
Jonathan Sillito (University of Calgary)

Centralized Version Control Systems (CVCS), such as CVS or Subversion, enforce having a single, central repository. Developers must have commit access to benefit from the support afforded by the CVCS. Decentralized VCSs (DVCS) such as Git, Mercurial, and Bazaar, relax this requirement. Each checkout from a DVCS is a first-class repository in its own right, providing equal revision control support to all developers, and not just to core committers. Here we present a simple scenario to compare how the type of VCS can impact how an outside contributor may contribute to an open-source project.

**Scenario:** Tom, an outside contributor, wishes to contribute a patch to an open-source project. His proposed change must meet with the approval of Sally, a core developer.

**Time**

- Tom checks out code
- Tom prototypes several approaches
- Another dev makes a commit to mainline
- Tom updates his patch against mainline
- Tom sends changes to Sally for review
- Sally examines Tom's patch
- Sally prototypes some suggested changes
- Tom continues to make changes in parallel
- Sally sends her changes to Tom as a patch
- Tom examines Sally's recommendations
- Tom adjusts code in light of recommendations
- Tom sends his final changes to Sally as a patch
- Sally approves and merges Tom's patch

**DVCS:** Although Tom is not a core committer on the project, Tom enjoys first-class support from the DVCS, with the ability to make commits as he sees fit. Because Tom's commits are private until shared, Tom can commit at opportune times, rather than only once he has ensured that all tests pass.

**CVCS:** Although Tom can checkout and update against the project mainline, he cannot use the project's VCS to help in managing his proposed changes, and must instead maintain his own system for tracking changes (e.g., a parallel CVCS, or through copies, such as zip files).

**DVCS:** Tom merges with the mainline and sends a rich patch to Sally for review; this richer patch includes the revisions the patch was made against. Sally would know if Tom had applied other patches that are not included in his own patch, or if his patches were not made against the current tip of the project mainline.

**CVCS:** Before creating a patch, Tom must ensure that his code is up to date with the mainline to ensure his patch will apply cleanly. Otherwise he must explicitly indicate the mainline revision that the patch was made against.

**DVCS:** Sally makes a private branch and merges Tom's patch to examine Tom's proposed changes.

**CVCS:** Sally checks out from the branch and applies Tom's patch. Although Sally has commit rights to the project VCS, creating a branch in a CVCS is typically considered to be a relatively heavy-weight operation, and is avoided for small one-off changes. Thus Sally must also manually manage changes.

**DVCS:** Sally merges Tom's change into the project mainline; this patch carries the full development history. CVCS: Typically patches from non-committers bear no trace of the development history of the patch, such as the interim commits.

**DVCS:** Tom merges Sally's changes with his other changes; although most CVCS support 3-way merges, neither the project VCS nor Tom's nor Sally's VCSs (if any) know of the other versions; Tom must manage the merge by hand.

**CVCS:** Tom must merge Sally's changes with his other changes; although most CVCS support 3-way merges, neither the project VCS nor Tom's nor Sally's VCSs (if any) know of the other versions; Tom must manage the merge by hand.

**Summary**

DVCSs have a number of advantages over traditional CVCSs: (i) DVCSs provide first-class revision control support to all developers, regardless of their commit privileges; (ii) DVCSs maintain revision metadata to enable simple, automatic, and repeated merging; (iii) DVCSs support easy experimentation by providing cheap branches; and (iv) DVCSs support disconnected operation.

Transitioning to a DVCSs is not a simple operation and may require substantial background work to adapt existing development practices and workflows. Repository metadata, such as a revision number, are widely used by developers and embedded in comments, bug reports, and mail messages. Since each checkout is a first-class repository, once code has been released can rarely be retrieved and removed.