Darcs, concepts and internals

Sebastian Fischer

Darcs is a distributed version control system written in Haskell. It's internal model is dual to the model of most other version control systems. Instead of tracking a tree (or, in case of merging, directed graph) of *states* of a repository, Darcs tracks *changes* which might (but need not) depend on each other.

The state of a Darcs repository is determined by the set of changes it contains. One advantage of this different mental model is that different lines of development can be merged more easily than in state based systems. No special merge patches are required for combining independent lines of development and it is simple to pick isolated changes without having to consider potential previous work of another development state as unnecessary dependency.

While the mental model of a Darcs repository is conceptually only a partially ordered set of changes, Darcs stores these changes internally in a specific order. It is hence a fundamental operation in Darcs to *reorder* independent changes. Together with a notion of *undoing* changes, reordering is the basis for implementing the commands Darcs provides to users for managing repositories. Merging different lines of development is one example that can be expressed in terms of undoing and reordering changes.

A disadvantage of a stateless view on repositories is that conflicts cannot be described and handled as easily in terms of differing states. Conflict handling has historically been a source for problems for Darcs's performance. Today, the situation has improved but possible inefficiencies remain. So called conflict fights, where conflict resolutions are applied only in one repository and repeatedly conflict with further developments pulled from a different repository, are still a source for exponential run time and users need to take care to avoid them.

Abother consequence of a mental model based on only partially ordered changes is that the notion of history is somewhat diffuse. Different repositories that are equivalent in terms of the overall changes they represent may store specific changes in differing orders internally. Users looking at the changes stored in such repositories will get output that differs with regard to the order in which changes are listed. Every recorded change has an associated date and author that lets users see who implemented a change when. But if a change originated

somewhere else it remains unlear who applied the change when to the current repository. Darcs has a post-hook mechanism that can be used to track such additional history information.

Understanding the difference between state based and change based views on repositories is important to make an informed choice between Darcs and other version control systems. Darcs model of changes that can be undone and reordered provides a base for its commands which emphasize conceptual simplicity and consistency.