

History reordering: performance and features

Student: Alejandro Gadea
Mentor: Guillaume Hoffmann

Abstract

The goal of this project is to improve history reordering and make the most of it. Automatic history reordering is an operation that is the basis of many commands of darcs. We want to take advantage of it by adding two features: a minimizing context feature for “darcs send”, and a dependency graph generation that would enable a third party software to visualize a darcs repository in a non-linear way. As a prelude to this work, we will measure and fix performance of automatic history reordering.

About the project

Darcs is a revision control system that presents a simple interface to the user, while automatically doing some operations that can reorder the history on demand. This enables it to make some operations like cherry-picking (transferring a single patch from a branch to another) more intuitive than with other systems.

Good performance is always something desirable in any software, in the particular case of Darcs, history reordering happens more often that one can imagine, for example making a “pull” to reclaim patches implies realign the inventory. The same happens with “push”. Thus, even the simplest use of darcs involves this operation.

The history of a darcs repository is, at the implementation level, an ordered sequence of patches. The recorded state of a repository is the result of applying these patches in this order to an empty state. Darcs is able to reorder a sequence of patches in a given history while maintaining the final recorded state of a repository. This is essential when comparing two repositories to transfer one patch from one to another.

Repositories can contain tags, that are special “empty” patches that represent a certain set of patches of a repositories. For instance, the user may create

tags that represent stable versions of a project (and call them 0.1, 0.2, 1.0, 2014-release, etc.).

When retrieving patches from a remote repository, darcs just puts them “on top” of the history of the current repository. With time, this can cause unnecessary divergences between various repositories, especially when tags are exchanged. For instance, consider the following repository with 3 patches:

$$[A, B, X]$$

If we retrieve patch C and tag T from another repository, with tag T tagging the set of patches $[A, B, C]$, our repository becomes:

$$[A, B, X, C, T]$$

In this history, the tag T is “dirty” because it has an unrelated patch (X) at the left of it. The command “optimize –reorder” reorders the history as follows:

$$[A, B, C, T, X]$$

The command “darcs optimize –reorder” must be manually run in order to “push” tags as early as possible in the history of a repository, thus avoiding “dirty tags”.

Reordering the history this way enhances performance of operations that involve comparing repositories; like pull, push and send. This is because comparing two repositories involves identifying the common part of both repositories up to the last tag, only if this tag is clean in both repositories. Furthermore, patch bundles created by “darcs send” (a command that generates patches as files to be sent by mail), are smaller when histories are reordered this way (patch files include some context up to the last clean patch).

Now, the command “darcs optimize –reorder” has a few shortcomings:

- It is abnormally slow in some cases. We even have cases where it seems to hang forever. We will collect and generate different test repositories to evaluate the performance of reordering. As well as measure it in time and in space (using profiling).
- When a repository has two tags such that neither is a superset of the other, “optimize –reorder” is not idempotent. That is, when this command is run once, one tag is brought up front, and when run again, the other one is brought up front. This seems to be a hole in the specification of the command, and could make two repositories look different while sharing the same history (up to reordering).

Furthermore, the idea is implement the flag “darcs send –minimal-context” that uses the algorithm to create patch bundles with as few patches as possible

in the context. The implementation will involve some heuristic-vs-exact thinking, since this task is computationally costful in general. We will see whether this option should be activated by default in every cases or only some of them.

Finally, we want to implement “darcs show deps”, a command that would export the dependency graph of patches of a repository, letting a third-party software display it. The history of a darcs repository is linear; contrarily to Git or Mercurial, there is no representation as a directed acyclic graph. However, we can construct a directed acyclic graph that represents the dependencies of patches between them. This construction involves exhaustive trying to reorder patches (to discover implicit dependencies between them). Then another software can display this graph.

Of course for performance reasons, this command should probably work by default up to the last tag of the repository. It should accept flags to dig up to the last N tags of the history, or completely.

The output format should be discussed. For testing reasons it will probably be simpler to output to Graphviz. But furthermore, outputting to a more widely used format like JSON should be done. This would for instance enable darcsden (the most advanced darcs repository hosting web engine) to display this graph from any repository.

Project Design

- Understand and document the current implementation of “optimize –reorder”
- Measure it in time and space consumption using real-world and script-generated repositories
- Thinking in the option –minimize-context, the option doesn’t exist for the command send. However having understood the algorithm of reordering seems a good step for the implementation.
- Implementing “darcs show deps”.

Timeline

- week 1: Understand the optimize –reorder implementation: collect examples (real-world and script-generated), profile, document (on wiki and code comments)
- week 2-4: Improve patch reordering implementation, documenting whether it should be idempotent.
- week 5-8: Implement darcs send –minimize-context.
- week 9-12: Implement “darcs show deps”

Benefits to the community

History reordering is omnipresent in darcs. Understanding and improving this operation will enhance the general experience of darcs. Moreover, reordering is involved in the calculation of the dependency graph of the patches of a repository, which is a long wanted feature.

Deliverables

- Faster “optimize –reorder”.
- “darcs send –minimize-context”.
- “darcs show deps” with at least one usable output format.

Challenges

Performance of history reordering is affected by patch-index, a cache data structure that should be updated each time history is changed. One way of dealing with this would be to measure performance without and with patch-index. If there are improvements that can be made to the maintenance of patch-index, we should look into it.

Responsibilities outside the project

As part of my Phd. i’m taking the course of Domain Theory that involves a “Take-Home” (not sure exact when) and the study and oral presentation of a paper related to the course in the last weeks. The course load is four hours per week initiating the 10 of march and finishing 19 of june.

About Me

I am Ale from Argentina, currently i’m a Phd. student in computer science. I have been working in haskell for the past 3 years in different open source projects from a group called Theona (★), the most interesting thing that I rescue from the experience in those 3 years is that working in haskell is fantastic and working in a friendly environment is the best of the best!

(★) For the curious, here are the main repositories:

Equ <https://github.com/alexgadea/equ-gui>

Fun <https://github.com/alexgadea/fun-gui>

Sat <https://github.com/manugunther/sat>

Hal <https://github.com/alexgadea/hal-gui>