Structured `diff` of C programs

Pierre-Évariste Dagand
pierre-evariste.dagand@lip6.fr
Cnrs – Inria Paris – Lip6

Abstract

This internship offers to design and implement a structure-aware `diff` tool for C programs. It will take place in the Whisper team of Inria Paris – LIP6, located at University Paris 6, and will be supervised by Pierre-Évariste Dagand (Cnrs).

The `diff` and `patch` programs figure predominantly in the programmer's toolbox. Given two text files, `diff` computes their line-by-line *difference*, *i.e.* it identifies the lines which were inserted, deleted or modified. Formally, it amounts to computing the longest common subsequence of two lists of strings [1, 2] or, dually, computing the edit distance [3, 4, 5]. Given a difference between two files, the `patch` program replicates its effect on the source input. Abiding by the Unix philosophy, these programs work on unstructured text files. As a result, they are efficient and widely applicable. However, on source code for instance, these tools cannot exploit the syntactic structure of programs to compute fine-grained changes, such as variable renaming or reordering of conditional statements.

With the advent of distributed version control systems (such as Git or Mercurial), the role of `patch` has been extended to encompass line-based *merging* of differences: in this context, developer Alice records her changes, which she sends to developer Bob who may have made local changes to the same file. `patch` must therefore apply the difference to a slightly different origin. Such an operation may fail if changes are *conflicting*. The granularity of the difference is crucial here: for example, if the unit of change is the whole line then independent edits on the same line will induce a (spurious) conflict. Distributed version control systems also had an influence on the programmers' workflow. For instance, it becomes humanly feasible to concurrently maintain several branches of the same software, each evolving at different pace. The Linux kernel thus have a main branch (`-next`) and a "long term" branch (`-stable`) to which only bug fixes are back-ported [6, 7]. However, these bug fixes may not apply directly: one then needs to modify them. In effect, the kernel developers end up *programming in patches*.

It thus becomes particularly tempting do compute differences on structured data [8] rather than mere lines of text. This idea has been applied in several tools, with some interesting large-scale applications [9, 10].

Internship objectives: Recently, colleagues at Utrecht University set out to develop a mathematical theory of differences and their merging semantics [11]. The goal of this project is to provide a canonical representation for differences and an abstract specification of the `patch` operation. This internship aims at exploring the impact of these early theoretical results on a concrete use-case: performing structured diff on C code. Our objective is two-fold: first, we wish to strengthen our confidence in the theoretical model by applying it on a non trivial example; second, we wish to guide the development of the theory through the constraints that arise in practice. In particular, we are interested in developing a theory that *scales by construction*. To this end, we should apply our methodology at the Linux kernel scale. We should also precisely characterize the complexity of our underlying algorithms. Depending on time and the student's interest, we would like to apply such a tool to implement a structured `blame` [12] and/or provide an interactive tool for programming in patches [13, 14].

Student's profile: We are looking for a student interested in algorithms, language design and mathematical abstractions (in particular, curiosity toward category theory). Acquaintance with a functional programming language (Scheme, OCaml, or Haskell) is strongly recommended. This work is funded by the Émergence(s) program of the City of Paris, thanks to which we can offer a stipend ("gratification") for the duration of the internship.

References

[1] James Wayne Hunt and Malcolm Douglas MacIlroy. *An algorithm for differential file comparison.* 1976. URL `http://www.cs.dartmouth.edu/~doug/diff.pdf`.

[2] Eugene W Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1 (1-4):251–266, 1986. doi:10.1007/BF01840446.

[3] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974. doi:10.1145/321796.321811.

[4] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64 (1):100 – 118, 1985. doi:10.1016/S0019-9958(85)80046-2.

[5] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33 (1):31–88, March 2001. doi:10.1145/375360.375365.

[6] Linux development team. `linux-stable`. URL `https://www.kernel.org/doc/Documentation/stable_kernel_rules.txt`.

[7] Jonathan Corbet. A discussion on stable kernel workflow issues, November 2016. URL `https://lwn.net/Articles/705220/`.

[8] Sudarshan S. Chawathe, Anand Rajaraman, Hector Garcia-Molina, and Jennifer Widom. Change detection in hierarchically structured information. SIGMOD '96, pages 493–504, 1996. doi:10.1145/233269.233366.

[9] Masatomo Hashimoto and Akira Mori. Diff/ts: A tool for fine-grained structural change analysis. WCRE '08, pages 279–288, 2008. doi:10.1109/WCRE.2008.44.

[10] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. Fine-grained and accurate source code differencing. ASE '14, pages 313–324, 2014. doi:10.1145/2642937.2642982.

[11] Victor Cacciari Miraldo and Wouter Swierstra. Structure-aware version control. IFL '16, 2016. URL `http://www.staff.science.uu.nl/~swier004/Publications/structure-aware-version-control.pdf`.

[12] Jake Edge. Token-based authorship information from git, August 2016. URL `https://lwn.net/Articles/698425/`.

[13] Andreas Grünbacher. How to survive with many patches or introduction to quilt, February 2012. URL `http://www.suse.de/~agruen/quilt.pdf`.

[14] Greg Kroah-Hartman. kernel maintainer's HOWTO for `quilt` and `-mm`, April 2005. URL `https://lwn.net/Articles/134072/`.