CSE 503: Project Proposal: Structured Delta

Instructed by *Michael Ernst*

Ji Luo (Student No. 1940053; UW NetID `luoji`)

16 April 2019

Problem Statement

Version control systems (VCS) are ubiquitously adopted in modern software development for codebase tracking. A popular VCS, Git, uses directed acyclic graphs to track versions of a repository, where each revision (commit) is a vertex with parent vertices pointing to it. Coding by a programmer corresponds to creating vertices with one parent – the parent represents the version on which ~~further coding~~ is based, and the new vertex represents the updated version. Git can display the difference (`diff`) between any two vertices. When the software is developed by a team, team members contribute to the codebase concurrently, resulting in a diverging graph (in contrast to a chain). It is thus important to be able to consolidate diverging revisions, a.k.a. merge the changes.

Merging and computing difference are closely related. A widely used strategy (if not the only one) for merging is to compute the difference between the two vertices to be merged with their nearest common ancestor, then combine the differences to produce the result. Git `diff`s text files by line (and binary files as a whole), which naturally leads to line-based merging strategy. The method takes advantage of the fact that code files are often formatted so that each line roughly corresponds to a fine-grained part in the semantical hierarchy (both for semantics of the programming language and semantics to human). However, this is far from being perfect. First, lines do not always correpond to part of the semantical hierarchy, as many programming languages ignore the difference between a whitespace character and consecutive whitespace characters, i.e., line separators can be inserted or removed without changing the semantics of the program. Insertion and removal of line separators are often done for aesthetic reasons (for example, to keep each line shorter than 80 characters). Second, lines can be too coarse for interpretation. For example, changing a subexpression inside a huge expression results in the whole line being identified as changed, which blurs the focus to that particular subexpression. Third, lines can be too fine for interpretation, which becomes a problem (only) when the formatting of a line is contextual. For example, a chunck of code accomplishes some task, which is later revised to be needed only when a condition holds. This chunck of code will be wrapped inside some `if` statement in the new version, thus having one more level of indentation. They will all be identified as changed under line-based comparison, bludged with the real change – a new precondition.

This creates (at least) two problems. One is that developers read the diff ~~ntiation~~ to *understand* the changes made between the revisions, and line-based `diff`s might blur and smudge the relevant, "real", changes with line artefacts. In fact, the interplay of having multiple changes inbetween might lead to less sensical `diff`s that forget the structure of the programming language (the change itself cannot be interpreted as well-structured non-terminals in the programming language). For example, accidental matching and mismatching of indentation levels of closing braces might result in a `diff` where logically grouped changes are displayed across scope boundaries. The other problem is that it naturally leads to bad merging results. See Figure 1 and Figure 2 for concrete examples.

Summary of Comments on luoji_3818302_55726627_proposal-comments.pdf

Page: 1

Number: 1 Author: mernst Subject: Highlight Date: 4/17/2019 12:53:27 PM
You don't need to say this. That guy isn't as important as the author of this document is.

Number: 2 Author: mernst Subject: Highlight Date: 4/17/2019 12:54:06 PM
VCS is singular. You mean "VCSes" here

Number: 3 Author: mernst Subject: Highlight Date: 4/17/2019 12:54:30 PM
This is true of every distributed VCS.

Number: 4 Author: mernst Subject: Highlight Date: 4/17/2019 12:55:12 PM
We often say "versions of the project". The repository is a collection of versions of the project.

Number: 5 Author: mernst Subject: Highlight Date: 4/17/2019 12:55:47 PM
It is more common that the node points to its parents, rather than the parents pointing to the node.

Number: 6 Author: mernst Subject: Inserted Text Date: 4/17/2019 12:57:13 PM
the change

Number: 7 Author: mernst Subject: Highlight Date: 4/17/2019 12:59:35 PM
Why is this important now? It seems unrelated to the curretn paragraph, which is about how the VCS stores the history.

Number: 8 Author: mernst Subject: Highlight Date: 4/17/2019 1:00:31 PM
The graph is not "diverging". (I'm not even sure what that term means.) Some nodes have multiple children or parents.

Number: 9 Author: mernst Subject: Highlight Date: 4/17/2019 12:59:58 PM
What does this term mean? Define your terms -- or even better, use standard terms.

Number: 10 Author: mernst Subject: Highlight Date: 4/17/2019 1:00:48 PM
I don't know what a "diverging" revision is.

Number: 11 Author: mernst Subject: Highlight Date: 4/17/2019 1:06:21 PM
This is not a good topic sentence for the paragraph. This is not what the whole paragraph is about. Split the paragraph into multiple paragraphs, or rewrite.

Number: 12 Author: mernst Subject: Highlight Date: 4/17/2019 1:07:35 PM
What does this mean (be precise, not vague), and why is it a problem?

Number: 13 Author: mernst Subject: Highlight Date: 4/17/2019 1:07:59 PM
Incorrect word.

Number: 14 Author: mernst Subject: Cross-Out Date: 4/17/2019 1:08:15 PM

Number: 15 Author: mernst Subject: Highlight Date: 4/17/2019 1:06:52 PM
I'm not sure what is the referent.

Number: 16 Author: mernst Subject: Highlight Date: 4/17/2019 1:08:34 PM
I don't know what this means.

Number: 17 Author: mernst Subject: Highlight Date: 4/17/2019 1:08:55 PM
A diff does not have a memory, and it cannot "forget".

```
1   1 #include<stdio.h>
2   2
3   3 int main(int argc, char **argv)
4   4 {
   -5     puts("Hello, world!");
   +5     if (argc == 1)
   +6     {
   +7         puts("Hello, world!");
   +8     }
6   9     return 0;
7  10 }
8  11
```

(a) A bad `diff`.

```
1   1 #include<stdio.h>
2   2
3   3 int main(int argc, char **argv)
4   4 {
   +5     if (argc == 1)
   +6     {
5   7         puts("Hello, world!");
   +8     }
6   9     return 0;
7  10 }
8  11
```

(b) A good `diff`.

Figure 1: Example exhibiting undesired result from line-based `diff`. Created with `https://strcmp.cc/`.

```
1   1 static void TimeoutFifoConnection()
2   2 {
3   3     Sleep(timeout);
   -4     if (!FifoConnected)
   +4     if (FifoConnected)
5   5     {
   -6         FifoStream fs = null;
   -7         try
   +6         return;
   +7     }
   +8     FifoStream fs = null;
   +9     try
  +10     {
  +11         if (direction == In)
8  12         {
   -9             if (direction == In)
  -10             {
  -11                 fs = OpenFifoAsReader();
  -12             }
  -13             else /* direction == Out */
  -14             {
  -15                 fs = OpenFifoAsWriter();
  -16             }
  +13             fs = OpenFifoAsReader();
17  14         }
  -18         catch
  +15         else /* direction == Out */
19  16         {
  +17             fs = OpenFifoAsWriter();
20  18         }
  -21         finally
  +19     }
  +20     catch
  +21     {
  +22     }
  +23     finally
  +24     {
  +25         if (fs != null)
22  26         {
  -23             if (fs != null)
  -24             {
  -25                 try { fs.Close(); } catch { }
  -26             }
  +27             try { fs.Close(); } catch { }
27  28         }
28  29     }
  -29     DeleteFifo();
30  30 }
31  31
```

(a) A bad `diff`.

```
1   1 static void TimeoutFifoConnection()
2   2 {
3   3     Sleep(timeout);
   -4     if (!FifoConnected)
   +4     if (FifoConnected)
5   5     {
   +6         return;
   +7     }
6   8     FifoStream fs = null;
7   9     try
8  10     {
                    ...
25  27             try { fs.Close(); } catch { }
26  28         }
27  29     }
  -28     }
  -29     DeleteFifo();
30  30 }
31  31
```

(b) A better `diff`.

Figure 2: Example exhibiting result that forgets language structure from line-based `diff`.
The change is from deep nesting to early return.
Note how the block delimiters for `if-else` and `try-catch-finally` are mingled.

Page: 2

Number: 1 Author: mernst Subject: Sticky Note Date: 4/17/2019 1:09:36 PM
These are good examples.

Vista to New Tool

Codes of most programming languages have a tree structure. Indeed, if the language is context-free and unambiguous, the parser already gives a unique parsing tree. Most programming languages are also insensitive to extra whitespace characters. With these combined, one can convert the code from textual representation to tree representation, track or find the differences between trees (instead of lines), and merge tree changes (instead of line changes).

The new tool to be developed should be capable to do the following:

- Compare two (sets of) code files and produce the difference (structured delta).
- Merge two sets of differences (perhaps only when both are created with the same reference) and deserialize the result back to text files.
- Output the merge operations in a easy-to-understand way so that a human can inspect the merging to avoid silent and wrong merge results.
- Have its components replaced (e.g., different parsers for different languages).

Additionally, it is desired that the tool be able to heuristically partition a set of differences into logically grouped subsets, which will provide more semantical output for human to *understand* the difference.

The tool should parse the text into trees and operate directly over the tree representation, without serializing the tree with some encoding, invoking external programs and reparse the output. Such chaining is not necessary and it is wasteful to have this detour. It is also expected that this tool performs well on home computers.

The project has the following novelties:

- It roots the problem of merging on `diff`ing, and extend from there on. The key observation is that once we have a good and well-structured `diff` result, we can do many things with it – exposition, merging, logical partitioning and perhaps more.
- It tries to improve efficiency by not using dedicated on-disk encoding.
- It tries to improve performance by operating over the trees (instead of conversion to and back from text).

Evaluation Criteria

The following metrics (compared with other tools) are of interest:

- The speed of producing the difference.
- The length of the difference (implicitly compared with the necessary length of the difference).
- The speed of merging two sets of difference.
- The consistency of automatic merging results with manual merging results (the quality or reliability).
- The frequency and the size of merging conflicts.

Page: 3

Number: 1 Author: mernst Subject: Highlight Date: 4/17/2019 1:09:25 PM
Wrong word choice.

Number: 2 Author: mernst Subject: Highlight Date: 4/17/2019 1:10:34 PM
Even if these are not the case, in practice the implementation still makes a choice.

Number: 3 Author: mernst Subject: Highlight Date: 4/17/2019 1:11:06 PM
Use first person only for tasks that the author performs manually. If a tool does the work, say that.

Number: 4 Author: mernst Subject: Highlight Date: 4/17/2019 1:11:59 PM
This sentence gives the key idea. You should communicate this in the first paragraph as well. Don't wait until halfway through the document to tell the reader.

Number: 5 Author: mernst Subject: Highlight Date: 4/17/2019 1:13:20 PM
I think this is not a new type of thing -- it is a regular diff, though created using a different algorithm. I would not give it a new name if it is the same as something that already exists. You can say that it is produced using a different algorithm, though.

Number: 6 Author: mernst Subject: Highlight Date: 4/17/2019 1:14:58 PM
Why do you here make this implementation detail explicit, whereas the previous bullet point did not?

Number: 7 Author: mernst Subject: Highlight Date: 4/17/2019 1:14:10 PM
What are "sets of differences"? Are they textual diffs?

Number: 8 Author: mernst Subject: Highlight Date: 4/17/2019 1:13:37 PM
What is a "referee"+

Number: 9 Author: mernst Subject: Highlight Date: 4/17/2019 1:15:20 PM
Can you describe this and/or show an example?

Number: 10 Author: mernst Subject: Highlight Date: 4/17/2019 1:16:05 PM
This is the first explicit mention that your tool uses a parser.

Number: 11 Author: mernst Subject: Highlight Date: 4/17/2019 1:17:33 PM
It is a good goal to design for this. However, I think it should be a stretch goal, but not a main goal, for the class project. I would rather that you produce a great product that only works for one programmnig language, rather than try a bigger project and not complete it.

Number: 12 Author: mernst Subject: Highlight Date: 4/17/2019 1:18:07 PM
If you do not have an explanation of why this is the case, and you are not explicitly testing this in your evaluation, you should not make the claim.

Number: 13 Author: mernst Subject: Highlight Date: 4/17/2019 1:18:50 PM
This paragraph is about implementation details. Move it elsewhere or omit it from the proposal (even though it will be relevant later).

Number: 14 Author: mernst Subject: Highlight Date: 4/17/2019 1:18:10 PM

Number: 15 Author: mernst Subject: Highlight Date: 4/17/2019 1:21:31 PM
This is a key difference from previous work. (By the way, it is essential that you cite previous work in your proposal! That will help readers to understand what is novel and how your tool performs differently from other tools.)

I think you are saying that you will (only) solve the problem of creating a better diff, and then you can use standard merge tools without change. Is that correct? If not, you need to clarify.

Number: 16 Author: mernst Subject: Highlight Date: 4/17/2019 1:22:11 PM
I think you must be thinking of some related work that uses on-disk encoding. But I don't know what you are thinking of, so this is obscure and confusing. Please clarify.

Number: 17 Author: mernst Subject: Highlight Date: 4/17/2019 1:23:23 PM
I don't think you should worry about performance. If the merge takes an extra second to perform, but 1% of the time it saves the programmer several minutes or more of confusing work, it is worthwhile. More generally, focus on the functionality first. Worry about optimization only if that turns out to be a real problem.

Number: 18 Author: mernst Subject: Highlight Date: 4/17/2019 1:23:52 PM
I don't think this is very interesting. It certainly shouldn't be the first criterion!

Number: 19 Author: mernst Subject: Highlight Date: 4/17/2019 1:24:02 PM
Same comment as above.

Comments from page 3 continued on next page

Vista to New Tool

Codes of most programming languages have a tree structure. Indeed, if the language is context-free and unambiguous, the parser already gives a unique parsing tree. Most programming languages are also insensitive to extra whitespace characters. With these combined, we can convert the code from textual representation to tree representation, track or find the differences between trees (instead of lines), and merge tree changes (instead of line changes).

The new tool to be developed should be capable to do the following:

- Compare two (sets of) code files and produce the difference (structured delta).
- Merge two sets of differences (perhaps only when both are created with the same referee) and serialize the result back to text files.
- Output the merge operations in a easy-to-understand way so that a human can inspect the merging to avoid silent and wrong merge results.
- Have its components replaced (e.g., different parsers for different languages).

Additionally, it is desired that the tool be able to heuristically partition a set of differences into logically grouped subsets, which will provide more semantical output for human to *understand* the difference.

The tool should parse the text into trees and operate directly over the tree representation, without serializing the tree with some encoding, invoking external programs and reparse the output. Such chaining is not necessary and it is wasteful to have this detour. It is also expected that this tool performs well on home computers.

The project has the following novelties:

- It roots the problem of merging on `diff`ing, and extend from there on. The key observation is that once we have a good and well-structured `diff` result, we can do many things with it – exposition, merging, logical partitioning and perhaps more.
- It tries to improve efficiency by not using dedicated on-disk encoding.
- It tries to improve performance by operating over the trees (instead of conversion to and back from text).

Evaluation Criteria

The following metrics (compared with other tools) are of interest:

- The speed of producing the difference.
- The length of the difference (implicitly compared with the necessary length of the difference).
- The speed of merging two sets of difference.
- 20 e consistency of automatic merging results with manual merging results (the quality or reliability).
- The frequency and the size of merging conflicts.

21

Number: 20 Author: mernst Subject: Highlight Date: 4/17/2019 1:24:22 PM

These are the most important criteria.

Number: 21 Author: mernst Subject: Sticky Note Date: 4/17/2019 1:25:48 PM

In addition to not discussing related work, you have not described the algorithm. It would be helpful to even say whether you will use tree merging algorithms (which ones) or whether you will depend on current merge algorithms that operate over diffs.