# Comparison of Tools for Digitally Tracking Changes in Text

Eve Washington[1], Bernat Ivancsics[1], Emily Sidnam-Mauch[2], Ayana Monroe[3], Kelly Caine[2], Susan E. McGregor[1]

Columbia University[1], Clemson University[2], The University of North Carolina at Chapel Hill[3]

Tracking changes in digital texts is a longstanding interface challenge, as early digital technologies left no recorded traces of alterations. Currently, two key categories of tools track text changes: code editing and word processing tools. Each has implemented different interface patterns to accomplish several goals: attributing change authorship, tracking the time of change, recording the change action taken, and specifying the location and content of the change. While some visual characteristics of change tracking are consistent across all tools, there are significant differences in change representation divided along the tool-type line, that may reflect their specific cultures of use. Overall, however, there is a limited range of visual methods for representing changes to digital text over time.



## INTRODUCTION

Tracking changes in digital text documents is an integral part of tasks ranging from collaboratively editing a scholarly document like this one to reviewing and editing code. Over the past four decades, the need to trace and attribute changes to a digital text has been concentrated in two primary areas of digital technology use: word processing and software development. Both have become increasingly collaborative and contested processes. Despite some similarities in overall task goals, approaches to visualizing changes to digital text documents vary. In this work, we survey the landscape of tools for digitally tracking changes in text and identify similarities and differences across tools.

## BACKGROUND

In collaborative document editing, displaying and communicating what parts of a document have been changed, when, and how is a need. While tools track this to a high level of granaularity have existed for decades, understanding the state of a whole document at a given point of time remains difficult (Viégas et al., 2004).

The history of tools for tracking and comparing versions of digital texts stretches back nearly half a century, with the introduction of the algorithm for the Unix `diff` command in 1976 (Hunt et al., 1976). Much of the work on this topic in the computer science community focuses on the algorithms for identifying differences between texts (e.g. Heckel, 1978; Miller & Myers, 1985; Tridgell & Mackerras, 1996). Much less emphasis has been placed on the visual display of those differences, although this capability is usually incorporated into various pieces of *version control software* (VCS) that incorporate `diff`-like features (Ruparelia, 2010).

Alongside these tools, which were largely developed by and for programmers using digital computers, is the evolution of digital word processing, which took place on standalone devices that were in use widespread use by the late 1970s and early 1980s (Kirschenbaum, 2016). As digital computers became more mainstream, dedicated word processors became obsolete and were replaced by computer software programs for word processing, many of which incorporated the visual editing conventions of typesetting, as demonstrated by the introduction of "revision marks" in Microsoft Word as early as 1987 (Inc, 1987). A new instance of text tracking developed upon the mass adoption of the internet and world wide web in the mid-1990s; wikis were developed through the crossover of code editing and word processing. These wikis applied version control methods and interfaces to non-code text documents (Viégas et al., 2004).

## METHODOLOGY

Through examining VCS and document review/track changes capabilities in word processing programs, our goal is to identify the user interface presentation mechanisms for the following elements in text tracking tools: 1) Authorship attribution; 2) Date/time of change; 3) Type of change (e.g. addition, deletion); 4) Substance of change (new text/old text); and 5) Location of change within the document.

To do this, we reviewed four instances of each type of software tool (VCSs and word processing programs). We selected our VCSs for review from a list generated as part of the 2021 Stack Overflow Developers Survey. Drawing on responses from 83,439 software developers from 181 different countries, the survey results show the most popular software development tools that included some form of version control display were: git, Visual Studio Code, and IntelliJ IDEA, each of which was used by 20% or more of respondents (*Stack Overflow Developer Survey 2021*, n.d.). We chose to complete this list with the popular code-sharing site GitHub, which is powered by git but provides a visual representation of codes changes, in contrast with git's native text-based approach.

Word processing software tools were selected based on their degree of difference from one another, in order to try to capture the widest variety of interface choices. By consulting blog posts discussing available tools (e.g., “9 Best Collaborative Document Editing Software in 2022,” 2020; Sha, 2021; Vigliarolo, 2020), we identified four distinct categories of word processing tools: wikis, paid-commercial, free-commercial, and open-source.We then selected the tools most frequently listed in these posts in each given category as the tool for our review.

After determining our sample, we reviewed documentation of the relevant track changes/versioning functionality, and evaluated our five elements of interest using

either visual references sourced from the program's official documentation or from a team member's installation instance.

## RESULTS

### Code Editors with Version Control Support

*git.* git is the core technology that drives many software version control packages but is also popularly used as a terminal or command-line utility during code development. Given its importance to the version control ecosystem and the ongoing popularity of its command-line interface, we chose to include the text-based representation of changes to files using the basic `git diff` operation as a baseline for our review.

As illustrated in Figure 1, git uses a line-based method to describe and visualize code changes. The first line shows the method of comparison (basic `git diff`) and the file name(s) - in this case, two different versions of the same file. The third line describes the "mode" and includes shortened hash of the files' respective contents, while the next two lines reiterate the filenames.

The fifth line begins the summary change description, in the form of the range of line numbers where differences have been found between the two files. In this case file `a` differs from file `b` from lines 1-5, and `b` is affected from lines 1-5. Following that is the actual substance of the change, in which shared lines begin with a space, added begin with a `+`, and removed lines begin with a `-`. In this default configuration (on ChromeOS), color is used to visually distinguish line ranges, file contents (which are subtly grey as opposed to white), removed text, and added text. The default behavior and usage of `git diff` output does not indicate authorship and/or date/time information ( *How to Read the Output from Git Diff?* , n.d.).

```
diff --git a/sample_text.py b/sample_text.py
index 13839c0..8878ce8 100644
--- a/sample_text.py
+++ b/sample_text.py
@@ -1,5 +1,5 @@
 Lorem ipsum dolor sit amet,
 consectetur adipiscing elit.

-Ut consequat,
-nunc dapibus feugiat vestibulum.
+Donec auctor,
+turpis nec semper luctus.
```

*Fig. 1: Output of a basic `git diff` command*

*GitHub.* Similar to git, GitHub observes text changes primarily at the line level, with additions and deletions color-coded and preceded by a +/- indicator; red represents deletions, green additions, and blue indicates ranges of non-differing content, described with the same line number convention as git. While inline modifications cause the whole line to be represented as either an addition or deletion (similar to git), GitHub provides an additional inline highlight to detail the substance of within-line text changes.


*Figure 2: Side-by-side presentation of changes in GitHub (Comparing Commits, n.d.)*

Unlike git, which can only present changes vertically on subsequent lines (Figure 1), GitHub offers the option to see changes presented side-by-side (Figure 2). In this view, changes are vertically aligned, and the distinct line numbers in each file/version are displayed. Keeping with local git implementations, GitHub only displays the author and the date/time of the more recent file in the basic file comparison view, though other views display the contributors to a file over time.

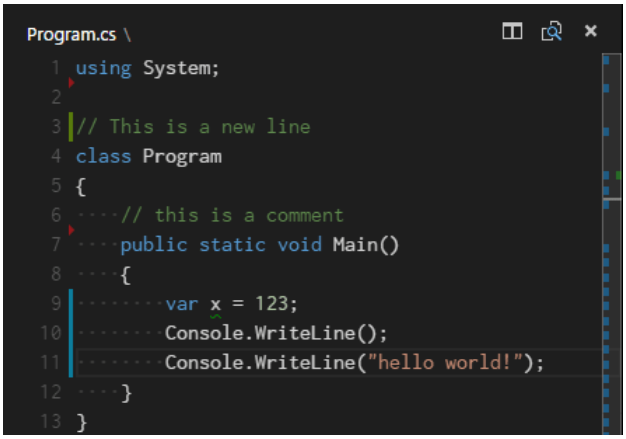


*Figure 3: Real-time changes visualized in VS Code (Version Control in Visual Studio Code, n.d.)*

*Visual Studio Code.* VS Code displays file changes in real-time during editing (Figure 3). Colored vertical margin bars indicate additions (green) and modifications (blue); red triangles mark deletions. Authorship/attribution and date/time information are not visualized in VS Code. VS Code also includes a `Source Control` tab that shares an interface with design choices almost identical to GitHub.

*IntelliJ IDEA.* Similar to VS Code, IntelliJ IDEA indicates editing changes in real-time using colors: green for additions, blue for modifications, and grey for deletions. These are displayed in the left margin while editing files and in a dedicated comparison tab while comparing saved versions.

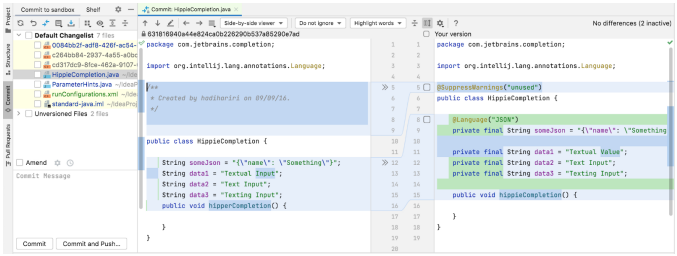

*Figure 4: IntelliJ IDEA commit, whitespace tracked (Review Changes | IntelliJ IDEA, n.d.)*

When comparing files using the 'Commit Tool', IntelliJ IDEA supports viewing changes side-by-side or vertically, similar to other tools. Uniquely, line numbers align in the side-by-side view, and, in both modes, users can customize the interface. For example, users can highlight whitespace changes, as shown in Figure 4, where the light blue indicates white space changes and movement of code snippets within the file *(Review Changes | IntelliJ IDEA, n.d.)*.

## Word Processing Tools

*Microsoft Word.* The track changes feature in Microsoft Word has two viewing options, `Simple Markup` and `All Markup.` The first uses red lines in the left margin to indicate where text changes have occurred and comments appear as speech bubble icons in the right margin, as shown in Figure 5.

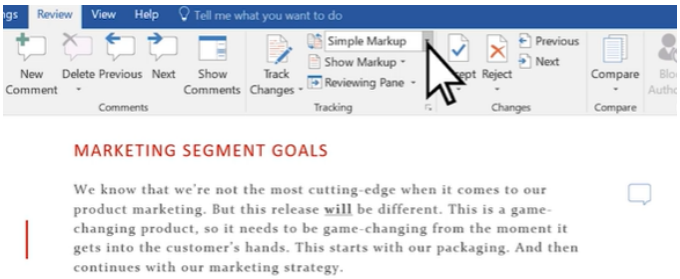


*Figure 5: Sample Simple Markup view in Microsoft Word (Track Changes in Word, n.d.)*

The `All Markup` view displays expanded and attributed comments in the right margin, in addition to a precise record of text changes and a leader line to their exact location within a line. However, users can extensively customize this interface, e.g., to use a combination of right-margin "speech bubbles" and inline edits. Users can select which types of changes they want to see (comments, insertions, deletions, or formatting) and from which users they want to see these changes *(Track Changes in Word, n.d.)*.

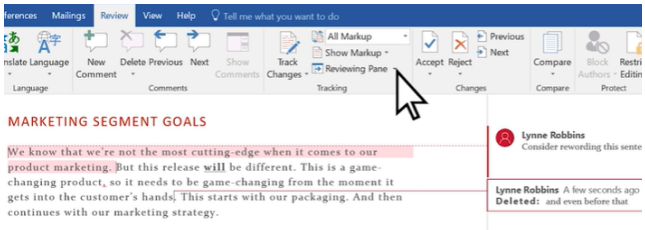


*Figure 6: Sample All Markup view in Microsoft Word (Track Changes in Word, n.d.)*

*Google Docs.* 'Suggestion Mode' in Google Docs tracks changes to files at the character level. Authorship (including username and photo icon, if available) and date/time information is automatically expanded in the right margin, aligned vertically with the edit. Inline, each user's changes appear in a distinct text color, along with a description of the change: additions, deletions, replacements, and formatting. Deletions and replacements are additionally shown inline, with the removed text displayed with strikethrough formatting, as shown in Figure 7. Similar to git and GitHub, Google Docs automatically records document-level versions by user and timestamp in a separate view.


*Figure 7: Google Docs 'Suggestion Mode'*

*Libre Office.* Though LibreOffice requires users to opt-in to change tracking via the 'record' method, users can choose from a variety of visual indicators. By default, added text is underlined in an alternate color, and removed text is shown in strikethrough. When users hover over a change, an attribution pop-up appears with any available authorship and date/time information. Like Word, left margin lines indicate which lines have been changed, as shown in Figure 8.

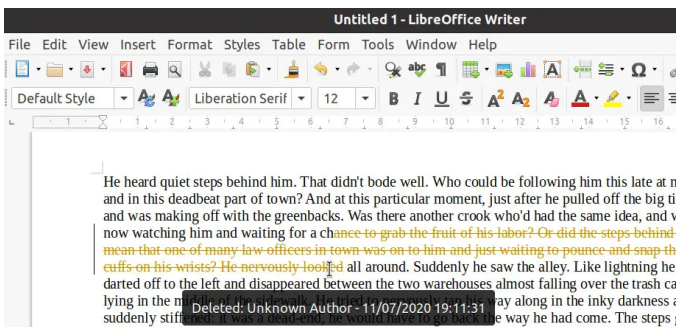


*Figure 8: Change and attribution information in LibreOffice (Track, Accept and Reject Changes in LibreOffice Writer, n.d.)*

Like Microsoft Word, LibreOffice allows both functional and aesthetic customizations of the visual interface. Text color can be used to distinguish among deletion, insertion, or formatting changes, assigned by user. Text formatting associated with the change type can also be customized, e.g., removed text can appear in bold instead of strikethrough. Users can also have changed lines indicated with highlighting, lines, or not at all (*Changes*, n.d.).

*Wikipedia.* Although Wikipedia content is similar to that typically edited via word processing software, its change tracking implementation is closer to that of code editors. Each "line" corresponds to a paragraph, and any change within a paragraph applies "change" formatting to the entire paragraph. Changes are shown side-by-side with a left-margin `-` and a yellow line indicating the previous version and a `+` and a blue line indicating the new version.

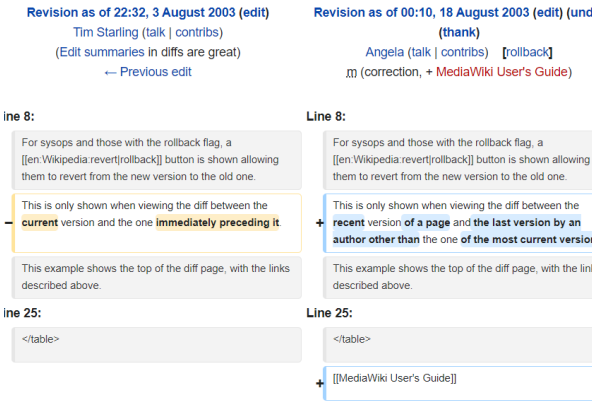


*Figure 9: Wikipedia Diff Viewer (Help:Diff - Wikipedia, n.d.)*

Changes within the paragraph are indicated in bold in the color of the border, and nearby paragraphs are displayed, shaded in grey, for context. If whole paragraphs are added or removed, the adjacent area appears blank, and no text formatting is applied, as shown in Figure 9 (*Help:Diff - Wikipedia*, n.d.).

Like git and other code editors, Wikipedia can only show differences between two versions of a page at a time, but authorship attribution and date/time of change is displayed above the change summary.

## DISCUSSION

Reviewing how text changes are represented visually across code editing and word processing tools, we found a set of common conventions within tool categories, as well as areas of crossover.

All tools make use of text color and formatting to visually indicate text additions, deletions, and replacements. At the same time, version control software shares more conventions across instances, using green/red for additions/deletions, and `-/+` to distinguish between original and updated contents. Word processing programs, by contrast, shared some defaults (e.g. strikethrough for deleted text) but support near-complete user customization. Across all the tools, we identified only four attributes that were modified; 1) Page annotation, 2) text color, 3) text highlight, and 4) text-decoration (strikethrough, underline). Different modifications of each of these attributes allowed for four different types of changes to be communicated; 1) Addition, 2) Deletion, 3) Inline modification, and 4) White Space change, for at least one tool in either category.

| | Page annotation | Text Color | Text highlight | Text decoration |
|---|---|---|---|---|
| Addition | Git<br>Github<br>Google Docs<br>VS Code | Git<br>Google Docs<br>Libre Office<br>Wikipedia<br>Word | Github<br>IntelliJ IDEA | Libre Office |
| Deletion | Git<br>Github<br>Google Docs<br>VS Code<br>Word | Git | Github | Google Docs<br>Libre Office<br>Wikipedia |
| Inline modification | Libre Office<br>VS Code<br>Word | | | |
| White Space | Google Docs | Github<br>IntelliJ<br>IDEA | | |

*Table 1: For all the observed types of changes, the visual indicators used to identify the change for each of the tools*

While both types of tool can track precise text changes, moreover, the unit of emphasis differs, with code editors emphasizing changes at the line level (with some further distinguishing inline differences), while word processing programs typically show character-level changes indicated by precise leader lines. Wikipedia represents a type of crossover, following code editor conventions at the paragraph level.

Other distinctions between the tool types further reflect their cultures of use, despite the fact that, as text editors, any of them could theoretically be used for either code writing or word processing. For example, both human programmers and computers read and refer to code by line number, making this type of reference especially relevant in version control software; this is less relevant in human-facing text documents. Likewise, computer programs are often intentionally modular, making the immediately surrounding code in a computer program less meaningful than it is in a written document; this may be why unchanged portions of code are typically collapsed in version control displays, but the entire text (including deleted elements) is often are displayed and attributed at a highly granular level in word processing documents.

In word processing, accuracy of language is often dependent on character-level distinctions, from conjugation to punctuation. Formatting and white space are similarly key to accurate interpretation and are therefore often given the same degree of emphasis as adding or removing characters. Semantic meaning in computer code, by contrast, is highly constrained by the language in use; the fact that only one

version control system clearly records whitespace reflects the fact that the vast majority of programming languages are whitespace-independent. Word processing tools also universally attributed authorship and date/time information to changes, while the code editors did not. This aligns with the idea that the *source* of an edit in word processing informs interpretations of the change, while other considerations (e.g. brevity, functionality) may be more important in code documents. Thus, word processing programs' change tracking emphasizes a collage of authorship over time, while code editors display differences between point-in-time "snapshots" that, on the surface, are attributed to a single user.

Research indicates that users prefer that similar tools use similar interface conventions (Experience, n.d.), so the shared visual representations of text changes within tool types is not surprising. In the case of version control systems, moreover, many internally rely on or at least natively support git, so their shared interface choices are understandable. In word processing, Microsoft Word predates all other tools evaluated, so competitor programs may attract users more easily by adopting its core conventions.

Still, five out of the eight tools we reviewed offered options for customization. For IntelliJ IDEA, Word, and Libre Office, for example, users can select the page layout of changes, but they also allow users to filter what types of changes are visible and how they are represented. These options offer a glimpse into the breadth options for visually representing changes in digital text, many of which are underexplored. Likewise, we note that Wikipedia remains the only substantive example of an effort to display change tracking on a large corpus of published, human-readable text, suggesting that there is opportunity for significant experimentation in this area, as, the overall range of change representations across all types of tools remains somewhat limited.

## LIMITATIONS AND FUTURE WORK

A major limitation of this work is the number of tools and options configurations we could feasibly review; as such, there may be variations we have missed in focusing on the most popular or best-known tools. A future, more comprehensive exploration may reveal more variation than presented here.

Another limitation is in our descriptive approach to visual analysis. Our current approach of summarizing user interface choices is sufficient to understand patterns at a high level, but a more substantive comparison would require a more formal and in-depth heuristic analysis, such as Nielsen's Usability Heuristics or another system of evaluation.

## ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1940670, Grant No. 1940679, Grant No. 1940713. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

9 Best Collaborative Document Editing Software in 2022. (2020, October 1). *Bit Blog*. https://blog.bit.ai/collaborative-document-editing-software/

*Changes*. (n.d.). Retrieved February 23, 2022, from https://help.libreoffice.org/latest/en-US/text/shared/optionen/01060600.html?&DbPAR=SHARED&System=WIN

*Comparing commits*. (n.d.). GitHub Docs. Retrieved February 23, 2022, from https://docs.github.com/en/pull-requests/committing-changes-to-your-project/viewing-and-comparing-commits/comparing-commits

Experience, W. L. in R.-B. U. (n.d.). *Jakob's Law of Internet User Experience (2 min. Video) (Video)*. Retrieved February 23, 2022, from https://www.nngroup.com/videos/jakobs-law-internet-ux

Heckel, P. (1978). A technique for isolating differences between files. *Communications of the ACM*, *21*(4), 264–268. https://doi.org/10.1145/359460.359467

*Help:Diff—Wikipedia*. (n.d.). Retrieved February 23, 2022, from https://en.wikipedia.org/wiki/Help:Diff#How_it_looks

*How to read the output from git diff?* (n.d.). Stack Overflow. Retrieved February 23, 2022, from https://stackoverflow.com/questions/2529441/how-to-read-the-output-from-git-diff

Hunt, J. W., McIlroy, M. D., & Bell Telephone Laboratories. (1976). *An algorithm for differential file comparison*. Bell Laboratories.

Inc, I. M. G. (1987). *InfoWorld*. InfoWorld Media Group, Inc.

*Incremental Diff – Sublime Text Documentation*. (n.d.). Retrieved February 23, 2022, from https://www.sublimetext.com/docs/incremental_diff.html

Kirschenbaum, M. G. (2016). *Track Changes: A Literary History of Word Processing*. Harvard University Press.

Miller, W., & Myers, E. W. (1985). A file comparison program. *Software: Practice and Experience*, *15*(11), 1025–1040. https://doi.org/10.1002/spe.4380151102

*Review changes | IntelliJ IDEA*. (n.d.). IntelliJ IDEA Help. Retrieved February 23, 2022, from https://www.jetbrains.com/help/idea/viewing-changes-information.html

Ruparelia, N. B. (2010). The history of version control. *ACM SIGSOFT Software Engineering Notes*, *35*(1), 5–9. https://doi.org/10.1145/1668862.1668876

Sha, A. (2021, February 26). *10 Best Free Word Processors You Can Use*. Beebom. https://beebom.com/best-free-word-processors/

*Snapshot*. (n.d.). Retrieved February 23, 2022, from https://docs.github.com/en/pull-requests/committing-changes-to-your-project/viewing-and-comparing-commits/comparing-commits

*Stack Overflow Developer Survey 2021*. (n.d.). Stack Overflow. Retrieved February 23, 2022, from https://insights.stackoverflow.com/survey/2021/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2021

*Track, Accept and Reject Changes in LibreOffice Writer*. (n.d.). Retrieved February 23, 2022, from https://www.libreofficehelp.com/track-changes-libreoffice-writer/

*Track changes in Word*. (n.d.). Retrieved February 23, 2022, from https://support.microsoft.com/en-us/office/track-changes-in-word-197ba630-0f5f-4a8e-9a77-3712475e806a

Tridgell, A., & Mackerras, P. (1996). The rsync algorithm. *Undefined*. https://www.semanticscholar.org/paper/The-rsync-algorithm-Tridgell-Mackerras/d9695436e01795fa572df1f01d8643056a96f205

*Version Control in Visual Studio Code*. (n.d.). Retrieved February 23, 2022, from https://code.visualstudio.com/docs/editor/versioncontrol

Viégas, F. B., Wattenberg, M., & Dave, K. (2004). Studying cooperation and conflict between authors with *history flow* visualizations. *Proceedings of the 2004 Conference on Human Factors in Computing Systems - CHI '04*, 575–582. https://doi.org/10.1145/985692.985765

Vigliarolo, B. (2020, November 5). *5 free alternatives to Microsoft Word*. TechRepublic. https://www.techrepublic.com/article/5-free-alternatives-to-microsoft-word/