Edward, edMUnD & Edwin: Line-Based Text Editing for the 21st Century

NATALIA POSTING
editor drone
equa.space

KATIE WOLFE
innocent bystander
katie.host

Abstract

In this paper we demonstrate how the standard text editor, ed(1), can be adapted into an IRC-based collaborative environment, paralleling modern IDEs such as Google Docs.

1 Introduction

Despite our best efforts, the human race still finds itself needing to edit computer files. The history of computer text editing began with *line editors*. Bound to the restriction of paper teletypes, such editors worked with entire lines at a time and, due to the low speed of teletypes of the time, only sent output when explicitly requested.

Ed(1) is such an editor. Originally designed for the Unix operating system in 1969, ed(1)'s influence runs deep in Unix tools such as grep and can still be traced in modern text editors such as ex. An annotated example of an ed(1) session follows.

```
 $ ed                                          Start ed(1).
→ e message.txt                                Open message.txt into the editing buffer.
← 59                                           The number of bytes read.
→ 1,$p                                         Print from the first line to the final line.
←                [03] DAYS
←        SINCE LAST TELETYPE INJURY
→ 1s/3/0/p                                     On line 1, replace 3 with 0, printing the fixed line.
←                [00] DAYS
→ $a                                           Enter input mode, appending after the last line.
→
→ note: rage-induced accidents not counted
→ quit                                         Oops!
→ arghahjvnfsjv
→ .                                            Exit input mode.
→ quit                                         Try to quit.
← ?                                            (Invalid command suffix.)
→ q                                            oh yeah
← ?                                            (Warning: buffer modified.)
→ q                                            Quit without saving.
```

Figure 1: A simple ed(1) session. Lines are first addressed, either by themselves or as ranges; then operations are applied to them; finally, a suffix such as p may be specified, causing ed(1) to print the line it operated on. Ed(1) is relatively quiet about all this: even its error messages just tell you that you goofed it.

Ed(1) remains powerful. Despite better judgement, the unit of the line serves as the foundation for almost every software engineering toolset, from the syntax of modern programming languages to the diff-tracking and conflict-resolution features of version control systems like Git. Code can be navigated in ed(1) using the indentation of source files as a guide to their structure; complex operations can be automated using regular expressions; and integration with language servers, linters, and build systems can be achieved with the shell command, `!`. But despite its enduring editing prowess, ed(1) is ultimately limited by the design of its original environment. Unix was a *time-sharing* operating system: users would pay a fixed fee for rights to use the machine's limited facilities during an allotted calendar period. Real-time collaboration was impossible.

Google Docs is an online, collaborative visual editor first released in 2006. Owing to the later invention of the color television (CRT), it allows for rich, colorful document formatting and provides live feedback the instant a character is typed. Furthermore, Docs allows for live collaboration and feedback between multiple users editing the same document. Its real-time iteration and collaboration capabilities have become a standard for enterprise software projects and landed Docs an easy position as the world's foremost IDE. Google has also supported being evil since 2018; while ed(1) supports basic evil, it is ultimately limited in scope to the design of its original environment.

Despite these limitations, we believe ed(1)'s line-based model can be adapted to match and even exceed the performance and feature set of modern IDEs such as Docs. In Section 2, we introduce our barebones prototype editor used to develop our later designs via the Internet Relay Chat and research editor interaction. In Section 3, we describe a failed but insightful initial attempt at modeling shared text files with a graph-like structure. In Section 4, we detail our final shared editing model, and in Section 5, we compare it to Docs and other modern development environments. We conclude in Section 6 with the potential of future work.

2 Edward

Edward is basically like this little guy. He dutifully sends lines back and forth between an ed(1) process and any number of IRC channels. As a direct mirror of ed(1), he doesn't distinguish between users or implement multi-buffer logic—every channel has one shared "head" controlled by all participants at once in a consensus model.

```
<natalia> edward: a
  <katie> edward: .
```

Figure 2: An example Edward session.

Edward's model limits the possibilities of collaboration but can still mirror the popular technique of pair programming. Since he only takes input when explicitly addressed, communication can be done out-of-band within the editor channel itself, allowing text-only collaboration without management of a separate chat window. This mechanism provides the foundation for an experimental code review technique in which code is reviewed live while being written. This provides a more theatrical review process and frees developers from having to read their own code later.

Overall, IRC is well-suited to interfacing with line-based programs. Special characters like the tab complicate input and display for most chat clients, but the IRC protocol itself has no problem handling them. Placing ed(1) in a networked environment also amplifies its previously bounded evil capabilities: with a cleverly crafted commmand, any user can cause Edward to send an exponentially increasing number of messages to any unsuspecting chat room.

3 edMUnD

Traditionally, ed(1)'s central data structure took the form of a doubly linked list of lines. This system models insertions and deletions well for a single user but has difficulty in the recovery of edit conflicts. Docs works well with a single shared file and per-character feedback, but an editor handling entire lines at a time is more likely to run into conflict. Our first approach involved a simple variation on the linked list: every line linked to its adjacent lines, but the links were not required to be bidirectional.

edMUnD *(*Editor-Drawn Multi-User Non-Euclidean Dungeon) was our implementation of this model. It had two commands in addition to those of ed(1). One would create a "branch," copying a block of code and creating unidirectional links back to its context in the main text file. Once ready to merge again, the second command would patch the links to point to the new text, "detaching" whatever was previously in its place.

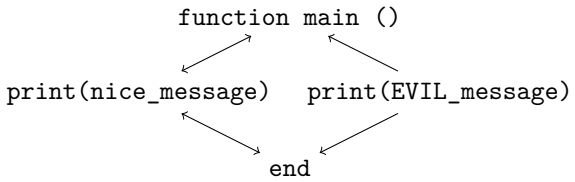

Figure 3: A new branch of code lurks, undetected.

Being able to continue editing in a detached state was useful: a user could work without interruption on code deleted by another, or save breaking changes in a hidden branch until later. It was also immensely evil: you could hide an ancient curse or a bad word and nobody would ever know.

These two basic operations provided more complex code layout as well. Simple combinations led to non-trivial results:

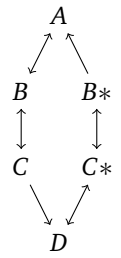

Figure 4: First, a new branch is created off of the middle two lines. Then only the latter line is merged back into the original layout. The result is a graph with no canonical representation: the text file reads completely differently depending on if it is viewed from the first line or the last.

The development of non-Euclidean text files showed exciting promise in the field of evil (think of inescapable textual labyrinths) as well as the field of interactive fiction (think of inescapable textual labyrinths). Unfortunately, the non-linearity of the files ultimately complicated PDF export, so the project was scrapped.

4 Edwin

Our final model, Edwin, uses a hybrid technique, combining the utility of edMUnD's branching mechanism and the stability of linear editing buffers. Inner details of the model and challenges faced in Edwin's development follow.

4.1 Buffers & lines

Edwin's primary organizational structure is the buffer. Each buffer is a self-consistent doubly linked list of lines. New buffers are created during branching operations, either explicitly via the branch command or implicitly by other editor operations (e.g. the deletion of a range of lines). Edwin has a trick up its sleeve that lets it parallel the smooth branching of edMUnD: every buffer contains links to the lines that were before and after it prior to branching. These links can be addressed directly or used in commands: the patch command, for example, attempts to place a buffer back into the original context of its old buffer. The explicit formalization of these boundaries saves the headache of accidentally creating paradoxical structures. Commands involving these patch links fail when the context is destoyed (i.e. if the lines around the original content are moved into different buffers and no longer have a path between them).

Lines in Edwin have unique identities; wherever possible, operations will retain them when modifying their contents or moving them around. This has the nice effect of retaining the positions of "heads" (representations of users in a channel), allowing text editing to go uninterrupted even if users remove or rearrange text in the middle of an operation.

4.2 Line numbering

The classic ed(1) provides one important mechanism for preventing destructive mistakes: line numbering. A cowardly user might wish to verify a range (e.g. `?^[fe]?,//`) is correct before deleting its contents; to check it, they can first use the `n` command to view the contents and line numbers of the addressed range, then use the line numbers directly to safely perform the operation. But in a collaborative environment, line numbers can silently change in between the two commands, leading to results as disastrous as getting the address wrong the first time. Edwin solves this by introducing two special line markers for every head called < and >, which always point to the first and last lines addressed in the previous command. Using `'<,'>` instead of line numbers, a user can address exactly the same lines as before.

4.3 History

Ed(1) implements one layer of editing history. Users can undo the last edit done and no more. Modern IDEs feature unbounded history and occasionally more complex time-traveling branch systems. Edwin compromises by implementing zero layers of history. Traditional undo methods were found to be complicated by the multitude of simultaneous edits in different locations. Rather, the function of an edit history is served flexibly by the message archive of the IRC channel itself, containing both edits and out-of-band annotations.

4.4 Extensibility

As part of a full Unix environment, ed(1) provides features external to the editor via a command which processes text through the shell. This is powerful in a Unix context but generally disconnected from the networking system Edwin resides in; instead, the recommended method for scripting Edwin takes place over IRC itself. Chat bots can interface with Edwin directly, inspecting data and modifying as appropriate. Given access to the editor command interface, IRC-based bots have *more* power than traditional shell scripts, as they can manipulate not only text but the state of the entire editor.

4.5 Incompleteness

The primary barrier to the use and analysis of Edwin is that I didn't finish implementing it. In order to continue research, we construct a model for what Edwin would look like and use a technique known as "guessing" to

generate precise thought experimental data. We avoid bias introduced by the variance in data collection by applying similar techniques to all test environments used in the editor's evaluation.

5 Comparison with other IDEs

How does Edwin fare against other development tools? We evaluate a set of editor environments according to four criteria: overall efficiency, appearance, synergy, and evil. As a baseline for Docs and Edwin, our primary contenders, we evaluate two classic editor environments: Vim over a paper teletype and Microsoft Paint over VNC.

5.1 Efficiency

While basic writing operations are simple in Paint, moving and replacing text requires manual intervention and is prone to failure due to Paint's relatively basic addressing system. All of these complications are amplified by the time required for a full screen refresh. Paint's ineptitude is only second to that of Vim, in which every keystroke sends dozens of mangled control characters to the poor, poor teletype output.

Docs does its job great: writing text has a noticeable network delay, but visual changes render fast on individual client machines. It supports a wide range of useful operations and has no problem restructuring large files. Edwin performs comparably if slightly better: feedback is only sent when explicitly requested, and moving from a granular editing system to a line-based one permits the correction of quick mistakes without network delays. Edwin's range of operations is similar to that of Docs, but is better inclined to complex organization with its native marks, buffers, and regexen.

5.2 Appearance

The appearance of code written in Paint is entirely dictated by its author. Paint has 24-bit color capability; its main limiting factor is the ability of the user to not just handwrite but handwrite *with a mouse over VNC*. Vim is somehow worse.

Docs has native rich text support, which permits users to highlight their code's syntax however appropriate. However, its font selection is limited: only the Google Fonts repository is available. Edwin supports rich text by means of IRC's formatting control characters and an unlimited range of fonts provided by chat clients.

5.3 Synergy

There is nothing more synergetic than a shared whiteboard. Paint is only limited by its single shared drawing cursor. Vim has no synergy at all—even if it were comprehensible, everyone'd have to crowd over the teletype. Might as well just draw on the paper.

Docs is at the forefront of channeling synergy through text: it supports the core components of group collaboration, with live editing, comment threads, and edit proposals. But it fails to recover synergy when it is lost—for example, when two simultaneous edits come into conflict, competing with one another for space in the finished document. Edwin supports all of Docs' features through a unified chat interface, and its branching mechanism allows for peaceful resolution of conflicting simultaneous work.

5.4 Evil

Paint supports evil: you can draw crude pictures wherever you want and, since its undo stack is limited, force your collaborators to either suffer through the drawings or completely erase whatever they intersected. Vim doesn't seem to support evil: certain Vim reimplementations in other environments are rumored to include it but were not considered for evaluation.

Google was initially reluctant to support being evil but faced pressure to allow large enterprises to use the quickly growing Docs; it eventually removed its evil restrictions in 2018. While now seeing broad use in evil, the editing interface is anything but: any malicious changes can be found and undone with its thorough history tracker, and drawing unsightly pictures in regular text documents is tedious. Edwin has supported evil since its conception, in both its use and its internal operation. It is easy to not only write swear words but irreparably overwrite entire documents, exponentially increase the editor's used memory, and cause the bot to flood any IRC channel naïve enough to associate with it.

6 Future work

I sure hope it d# 7 Future work

It would be nice if the editor existed. Additionally, Edwin's realization lacks a system for traditional, long-term collaboration. One potential model would use email: users could send files containing Edwin commands to a project maintainer, who would apply the commands to a central repository.

References

[1] IEEE and The Open Group. 2018. ed – edit text. The Open Base Specifications Issue 7, 2018 edition. `https://pubs.opengroup.org/onlinepubs/9699919799/utilities/ed.html`

w
quit
q
^C