# What’s the Diff? Version Control and Revision Reflections

*“Version control. Always. Everywhere. For everything.”   
– Karl Stolley, “The Lo-Fi Manifesto 2.0”*

Do you ever wonder what you’re supposed to learn in a writing class? And please don’t just say “to write”: if there’s one thing we’ve found in the last sixty years or so of research on writing, it’s that writing situations are incredibly varied; there is no one thing called “how to write” that anyone can pick up in 15 weeks, or a year. Dr. E. Shelley Reid, a stand-out teacher of writing teachers, has made the case that we’re really working on *how to deal with writing problems*: how to identify them, some strategies to try for the ones we know, and the confidence to keep trying even if a problem hasn’t gone away. Because the truth is, even expert, professional writers face writing problems. (And solving one problem often introduces a new one.) It’s just that some people have more practice in surmounting them.

I’m not going to recap Reid’s whole argument (though I do highly recommend her *Writing Spaces* essay on “Ten Metaphors for Writing”). Suffice it to say, one skillset you should build over the course of a writing class is a widening set of *revision strategies*: different ways to step into a draft and take it to the next level. In this essay, I’ll argue that studying the differences between drafts—your revision histories, or diffs—is a great way to find out what revision strategies you’re already using, and which ones you have yet to try.

I’m taking the phrase “revision strategies” from one of the classic studies of writing processes, by Nancy Sommers. By comparing early and late drafts, she found that beginning student writers tended to make changes at the level of word, phrase, and sentence, and that most of the word/phrase changes were substitutions: they didn’t change the overall structure or meaning. Experienced adult writers made those changes, too, but also tended to go beyond—they made more changes at larger-than-paragraph levels, like theme or section, and they did a lot more cutting and reordering.

There’s more to it than the mechanics: the differences here point to different understandings of why we’re revisiting the piece in the first place. If we’re mostly swapping out one synonym for another, then the problem we’re solving is probably about polish and presentation. That approach can be useful for adapting to or shaping our audience, and so I’d never say that changes smaller than the sentence aren’t important—but there are other problems that revision can address. Movement often signals deeper thinking, or re-thinking, because it’s a reconfiguration of what we thought we were doing in the earlier draft. For example, we might realize that an idea we’d come to only at the end of a paragraph or page really ought to come earlier (reordering); in that new position it may trigger new connections (addition in the middle of the draft, not just at the end). Whole paragraphs that lead in different directions might now cease to be relevant (subtraction), or might need new transitions to make sense (substitution of whole sentences, not just phrases).

Where do you find yourself on that spectrum? When you have to revisit a first draft, do you look for ways to “clean it up”? Or do you ask yourself what else, what next idea or better explanation, the draft helps you figure out?

What do you think your revision history would have to say about it?

## Making Change Visible

There are lots of ways to see what’s changed between drafts. Both Microsoft Word and LibreOffice have built-in functions to compare two documents, which will highlight new or changed lines and show deleted ones in strikethrough format. In recent versions of Word, you can even selectively focus on non-formatting changes (or, if you prefer, solely formatting changes). One nice thing about this feature is that it can be purely retroactive. That is, you don’t need to plan ahead for it, so long as you have two different files: a first submission vs. a final draft, say. (I’ll confess I’ve sometimes used this tool to see what’s changed in subsequent drafts turned in to me, both as a teacher and as an editor, even when the authors didn’t make a point of doing so for themselves. If you expect that your teachers or editors might do the same, you might as well get out ahead of them and see what they’re going to see.)

Even if this is all you take away from this essay, that may be enough to repeat the Sommers experiment for yourself. Compare two drafts of something you wrote before your writing course: what kinds of changes did you make, and where? Try it again at the end (or even the midpoint) of a writing course: how have your changes changed?

One potential problem with the compare-two-files approach, though, is that a lot of changes happen *between* submissions. If you’re only saving a file as “draft” and “final,” there’s a good chance you’re saving over the many revision steps you took in the process of getting to the version you turn in or send out. You could try to get around this by saving lots of interim drafts under separate names, but that makes for a lot of clutter in your file system—I think my 10-page dissertation proposal got up to something like “draft 23” before I finally submitted it—on top of which, all that duplication starts to eat away at your hard drive space.

A more robust solution is a *version control system*, or VCS. A VCS is software specifically designed to efficiently store and display the whole history of a project, including the diffs between points in that history. You get to keep the same filename throughout. (Or, if you change the filename, that change is itself stored as a diff.) If you’ve written with Google Docs, you’ve had one kind of VCS at work: it saves changes every time you pause, down to the second, and lets you browse through those changes. (Look for Version History under the File menu.) Wikis, too, store a history of each edit, in part so editors can recover text that shouldn’t have been deleted. (And isn’t it easier to make our own hard cuts when you know you have that safety net just in case?)

The terminology of version “control” comes from computer programming, where it’s a mainstay of collaborative projects that continue to evolve after they’ve been published. The VCS helps contributors get up to speed on what’s happened before, and prevents them from haphazardly overwriting each other’s work. But version control is just as helpful for a solo project. When I’ve had to spend some time away from a piece of writing, flipping through the project history helps me reacquaint myself with the directions I’d been heading in, recovering momentum in a different way than just reading through the current draft can offer.

For writers who use Google Docs, there’s even a browser extension that helps you with that recap. Called Draftback (http://draftback.com/), it gathers up all your revisions, from the smallest deletion of an extra space to the largest chunk of new text, and plays them back as a movie. As their promotional copy puts it, the effect is “like going back in time to look over your own shoulder as you write.” Dr. Elizabeth Chamberlain, a scholar of digital writing practices, has written that using Draftback helps her students reflect on their writerly choices more vividly and concretely. (If you haven’t been asked by your writing teacher to reflect on a project, you’ll almost certainly be asked soon.)

One thing Draftback makes especially clear is that for most people, no matter the number of revision moves in your repertoire, they don’t tend to happen in a set order: it’s not all the new text first, then all the editing later. Even individual sentences are usually written by a looping path, five words added in and two words taken out, revised on the fly as the thought comes clearer when you’re trying to express it. Not every sentence is hard-won like this; sometimes we can get into a flow and think in whole paragraphs at a time. But if you look at enough diffs, at a fine enough scale, I suspect you’ll find that even your “first thoughts” are revised thoughts. And that difference from a “first thought, best thought” mindset should free you to revisit and rethink even later on.

I noticed something curious in looking through my own Draftbacks. Places where things got tricky would play out for several seconds, long enough to read them, but major chunks of text would appear or disappear during playback in the blink of an eye. Why? It turns out every edit—whether a word or a paragraph—is treated as one change, and given equal time on screen. Because it doesn’t play the process back in real time (or else we’d be watching it for days, weeks, or more!), Draftback doesn’t *pause.* Where it seems to do so, it’s really showing multiple changes concentrated in the same part of the text. This is great for identifying stuckpoints, which can certainly be helpful to reflect on—I realized, for example, that I was often working and reworking the transitions between paragraphs in those moments of lingering diffs. But it actually makes it hard to see the breakthroughs, which are just as important, if not more.

This is why, as great as Google Docs can be, I started using something else for my own files: git, GitHub, and especially GitHub Desktop.

## Making a Commitment to Save Reflectively

Remember when I said that version control is a mainstay of collaborative computer programming? This is the ecosystem designed especially for that purpose: *git* is a lightweight program that keeps track of changes across all the files within a given folder on your computer; *GitHub* is a website that can synchronize or merge files and revision histories between git-tracked folders on your local computer and a centralized version online; and *GitHub Desktop* is a user-friendly program that makes it easy to use both. (Or not: GH Desktop also works just fine without sharing anything to the cloud.)

What makes this system different is that every change must be labeled in order to be logged: the system prompts you to write a short summary statement before it will enter it in the history. Git calls these “commit messages,” and the points in history “commits.” I kind of love the energy of that. It’s like, “Okay, I know you’ve saved this file, but are you ready to commit to it? Is this an official version you’d want to look at again later?” Pausing to commit—even to decide *whether* to commit—opens up a space for reflection *in the middle of the writing process*. Committing asks me to record not only what’s changed, but also what the goal of changing it was. Commit messages can say what’s new, or what you’re working on, or even what you’re planning to do in the future… or all three. (The structure of a commit message in git is one short required header line, then as much optional text as you want below that.) For my money, those pauses are where a lot of learning about revision can happen.

To help you decide what to write, or whether to go back and make more changes before you commit, GitHub Desktop’s main view shows the status of your changes: how many files are different from the previous commit, and, for simple text or image files, exactly what has changed.[1] Having the diffs right there helps me write the commit message in light of the specific changes, and if I forget to commit at the end of a working session, it reminds me of what I was in the middle of. (I’ve taken advantage of that feature several times in the course of writing this.)

Two important caveats, though. First, a git-based workflow encourages a bit of reflection, but it doesn’t enforce one. It’s entirely possible to write the required commit messages in a vacuous way, “draft 1,” “draft 2,” that don’t help you at all to see what’s changing when you glance down the revision history. You have to actually look to see what’s actually changed, what’s added, what’s taken away, or if you’re just refining in place. (And then to decide whether that’s what you wanted to be doing at that moment.) The reflection isn’t automatic.

Second, the save itself isn’t automatic, either: it’s entirely possible to make no commits at all between the first and final drafts. Until you get in the habit of writing a commit message every time you save (or stop working), it can be easy to forget. You’d end up with the same truncated sense of the file’s history as in the initial non-VCS scenario. Then not only would you lose the traces of all your diffs, you’d also lose text included in a middle draft but cut before the final. (That cut material might still be useful in the future, for a different audience or a different purpose. The time machine that version control offers you the chance to retrieve those interim ideas.)

Wouldn’t it be nice, then, if there were a way to combine Google Docs’ frequent automatic versioning with git’s moments of midstream reflection? In a way, there is: since 2017, Google Docs has allowed users to name particular versions, and to filter the history to show only named versions. You only get 80 characters, not multiple sentences, and you won’t be able to see the diffs as you’re deciding on a name. (This may lend itself more to descriptions of status, like “almost ready for peer review,” than to descriptions of changes, but you can still try for something closer to “improved transition from source A to source B.”) As compromises go, it’s not a bad deal.

## To sum up

It’s not always easy, in the thick of the writing or after pushing through the thicket, to remember what turns you took, or why; sometimes the new versions just replace what you’d done before, whether figuratively in your memory or literally on your hard drive. Using version control technology can help you track and make visible what you’re doing throughout the course of a writing project, so you can then assess how your strategies have changed—and where they might be useful again in the future.

## Notes

1. Note the caveat there: you won’t be able to see the latest diffs of a Microsoft Word file. (This is in part because Word files aren’t just text: they’re secretly a whole bunch of files compressed together. Try changing the extension from .docx to .zip and extracting to see what I mean.) But Markdown files, I’ve found, are pretty great for many of my writerly purposes, and work quite smoothly with all kinds of version control. And even if you’re not ready to go that lo-fi, you can still write commit messages to help you find specific versions to go back and compare within Word. A good VCS will help you recover the files at those past points in time.

## Works Cited