Version control as narrative device

Ben Klemens

1 August 2016

I'm a convert. I went from not getting distributed version control systems (DVCSes) to writing a book chapter on one (entry #187) and forcing it on my friends and coworkers. I originally thought it was about mechanics like easy merging and copying revisions, but have come to realize that its benefits are literary, not mechanical. This matters in how DVCSes are taught and understood.

The competition One way to see this is to consider the differences among the many distributed version control systems we aren't using.

Off the top of my head, there's Git (by the guy who wrote Linux, for maintaining Linux), mercurial (adopted as the Python DVCS), fossil (by the guy who wrote SQLite, for SQLite development), bazaar (by GNU to support GNU projects). So, first, in terms of who 'invented' DVCS, it seems like everybody saw the same problems with existing version control systems at about the same time, and developed responses at about the same time. All of these are big ticket projects that have a large network and geek mindshare.

But at this point, Git won the popularity contest. Why?

It's not the syntax, which is famously unclear and ad hoc. Want to set up a new branch based on your current work? You might start typing `git branch`, but you're better off using `git checkout -b`. There's a command to regraft a branch onto a different base on the parent branch, `git rebase`, which is of course what you would use if you want to squash two commits into one.

Did you stage a file to be committed but want to unstage it? You mean

```
git reset HEAD −− fileA
```

The git manual advises[1] that you set up an alias for this:

```
git config −−global alias.unstage 'reset HEAD −−'

#and now just use

git unstage fileA
```

I find it telling that the maintainers decided to put these instructions in the manual, instead of just adding a frickin' unstage command.

[1]https://git-scm.com/book/en/v2/Git-Basics-Git-Aliases

Yes, there are front-ends that hide most of this, the most popular being a web site and service named github.com[2], but it's a clear barrier that anything one step beyond what the web interfaces do will be awkward.

Early on, git also had some unique technical annoyances—maybe you remember having to call `git gc` manually. It has a concept, the *index* that other systems don't even bother with.

All that said, it won.

The index and commit squashing To go into further detail on the index that most DVCSes don't bother with, it is a staging area describing what your next commit will look like. Other systems simplify by just taking all changes in your working directory and bundling them into a snapshot. This is how I work about 98% of the time.

The other 2%, I have a few different things going on, say adding some text on two different parts of a document, and for whatever reason, I want to split them into two commits. The index lets me add one section, commit, then add the next and produce a second commit.

By splitting the mess of work into one item with one intent and a second with a new intent, I've imposed a narrative structure on my writing process. Each commit has some small intent attached, and I even have the ability to rearrange those intents to form a more coherent story. Did I know what I was doing and where I was going before I got there? Who knows, but it sure looks like I did in the commit log.

There's some debate about whether these rewritings are, to use a short and simple word, lying. The author of Fossil says[3] (§3.7) that not being able to rewrite the history of the project development is a feature, because it facilitates auditing. You can find others who recommend against using git rebase for similar reasons. On the other side, there are people who insist[4] on using git rebase, because it is rude to colleagues to not clean up after yourself. [And I stumble across tutorials on how to remove accidentally-committed passwords often.]

One side of the debate is generating an episodic narrative of how the project formed; the other is generating a series of news bulletins.

Personally, I am on the side of the episodic narrative writers. If you could install a keylogger that broadcasts your work, typos and all, would you do it? Would you argue that it is a matter of professional ethics that not doing so and only presenting your cleaned-up work is misleading to your coworkers?

Providing a means of squashing together the error and its fix loses the fact that the author made and caught a mistake, but that makes life more pleasant for the author and has no serious cost to others. Most information is not newsworthy.

You might draw the line in my ethical what-if at publication: once your work is out the door, correcting it should be accompanied by a statement. Git semi-enforces this by making it almost impossible to rebase after pushing to another repository. This seems to be more technical happenstance than a design decision.

[2] https://github.com

[3] https://www.fossil-scm.org/index.html/doc/trunk/www/fossil-v-git.wiki

[4] https://medium.com/@sellarafaeli/we-use-git-rebase-and-so-should-you-be89d1932a14#.mb0mumimj

On a social scale, revision control is again about building a narrative. Before, we were a bunch of kids running around the field kicking a ball around, and now we're a bunch of kids playing kickball. We don't throw tarballs or semi-functional patches at each other; we send discrete commits to discrete locations under specific rules, and use those to understand what colleagues were doing. We can read their commit history to know what they were doing, because they had tools to make the commit history a structured narrative.

I'd love to see somebody take my claim that revision control develops a narrative literally and actually use a DVCS to write literature. We are all familiar with interactive fiction[5]; why would revision control fiction, with its built-in sense of time and branching, be so unusual?

The evangelical disconnect This is the setup for so many awkward conversations. The evangelist has such zeal because of the structuring of unstructred work facilitated by the complicated features like the index and the rebase command; the newbie just wants to save snapshots. The evangelist may not be able to articulate his or her side, or the evangelist does talk about building a narrative, but a lot of people just do not care and see it as useless bureauracy, or don't have a concept of what it means until they are on the other side and have had a chance to try it themselves. I'd be interested to see tutorials that focus on the process of narrative writing, instead of rehashing the basics that are already covered so well.

In other evangelist news, I've saved you hours reading the million Git vs Other DVCS blog entries out there. The other one is easier to use, because it has fewer narrative control features.

If you buy my claim that Git's primary benefit over near-equivalent competitors is tools to rewrite the narrative, then Git's winning over alternatives is a strong collective revealed preference. Some of these same people who reject some programming languages because the semicolons look cluttery enthusiastically embrace a system where `git symbolic-ref HEAD refs/heads/baseline` is a reasonable thing to type. [Can you work out what it does and why I needed it?] That's a strong indication of just how high-value narrative control is among geekdom.

Of course, not everybody in a crowd is of like mind. DVCSes are socially interesting because they are something that working adults have to learn, with a real payoff after the learner gets over the hump. Like double-entry bookkeeping or playing the piano, the people who have taken the time to learn have a new means of understanding the world and generating narratives that people pre-hump may not see. It's interesting to see who has the desire to pursue that goal to its conclusion and who gives up after seeing the syntactic mess.

[5]`https://texturewriter.com`