Past Versions | Coursera

Past Versions

Past Versions

The basic principle underlying most revision control systems is that you *commit* your work into the revision control system, and it then keeps a snapshot of that version of your work. When you commit, you write a log entry, which describes what you have done. Later, you can review the log and return to an older version of your work if you need to. For example, if you decide to rewrite a piece of code to improve it, but find out that you have instead broken your program, you can return to a prior working version.

A "novice tools" approach to this problem would be to keep copies of your work and manage them by hand. You might think "well I could just copy all my code before I start, then use that copy later if I need to." If you are consciously aware that you are about to undertake a risky code rewrite, you might take this course of action. However, what if you start modifying your code and only later realize the trouble? If you use a revision control system, you should commit regularly, and thus will have good revisions in the past. Furthermore, if you make regular copies manually, you will find it hard to manage them all by hand.

Revision systems such as Git also have excellent tools for working with past versions. For example, Git has a command called **bisect**, which lets you search for where in the past you broke something. Suppose that I know that a feature was working 6 months ago; however, today I ran some tests and realized the feature was broken. I would like to go back and find exactly which version broke the feature, see what changed in that version, and then correct the code. The bisect command asks Git to help me search for the first broken version.

In particular, bisect binary searches through the revisions (thus the name), to find where the problem occured. You can either guide the search manually by telling Git whether each revision that it visits is good or bad, or you can have Git perform the search fully automatically by providing a shell script that determines if the revision is correct or broken.

Revision control is not limited to code, and in fact, is incredibly useful for any large, collaborative project (or even for just managing your own data on a small scale). To provide a concrete example of bisect, we use Git to revision control all of the materials for this specialization—LaTeX source, code examples, animations, and video recordings. One of the animations got inadvertently deleted, and we needed to restore it from the most recent version. Writing a shell script to test if that file existed, then running Git bisect found the revision where the file was deleted in a matter of seconds and let us restore it from the previous revision (and be sure we had the most up-to-date version!).

✓

Completed