AFIO: Inside an open source project

Koen Holtman

What do people actually do when they work in an open source project? What is the software process? Below I try to answer such questions by describing one particular case: my own work on afio, an open source archiver program that was initially created in 1985, and for which I have been the maintainer since 1993.

Introduction

While a lot has been written about open source software, much less has been written about the process of creating open source software. I know of a few good general accounts, which I refer to at the end of this article. In this article I will not describe the 'typical' or 'average' open source software process. What is average is a difficult question anyway, and depends in part on how broadly you define open source. Here, I give an account of my own activities in doing open source. I focus in particular on the case of the afio program. By using a specific case I can describe details of the micro-process that are often not covered in the more general accounts of open source development.

The afio archiver

Afio is a Linux/Unix program for packing up files into archives, and writing these archives to devices. It is very similar in function to the Unix 'tar' and 'cpio' commands, and the pkzip package in MS-DOS/Windows. Figure 1 shows the 'official' short description of afio that I bundle with releases.

> *Archiver & backup program with builtin compression Afio makes cpio-format archives. Afio can make compressed archives that are much safer than compressed tar or cpio archives. Afio is best used as an 'archive engine' in a backup script.*

Figure 1: The short description of afio in its Linux Software Map entry

The main attraction of afio, over the better-known tar, is that afio makes compressed archives in a safer way: it compresses the individual files in the archive, rather than the complete archive byte stream like tar does. If a compressed tar archive encounters even a single byte error on reading, the remainder of the archive cannot be unpacked anymore, and all the data in it is lost. Afio archives are more fault tolerant: a read error will generally only affect the unpacking of a single file.

Early history of afio

The first version of afio was written by Mark Brukhartz in 1985, or possibly even a few years earlier. I never talked to Mark Brukhartz, so I do not know exactly why he started afio. My guess, from the documentation he included, is that he needed a better version of the cpio program which he was using, and that he did not have the cpio source. In terms of software years, afio is ancient, and that is part of the attraction in maintaining it. They really wrote C differently back then. This is a typical fragment of the 1985 afio code, which is still part of the source today.

```
/*
 * inavail()
 *
 * Index available input data within the
 * buffer. Stores a pointer to the data
 * and its length in given locations.
 * Returns zero with valid data, -1 if
```

```
 * unreadable portions were replaced with
 * nulls.
 */
STATIC int
inavail (bufp, lenp)
     reg char **bufp;
     uint *lenp;
{
  reg uint have;
  reg int corrupt = 0;

  while ((have = bufend - bufidx) == 0)
    corrupt |= infill ();
  *bufp = bufidx;
  *lenp = have;
  return (corrupt);
}
```

In 1985, Mark Brukhartz added an open-source type license at the start of the afio source, and distributed it to others in the Unix community. I don't know exactly what distribution mechanism he used, but it was not FTP on the early Internet. In 1991, someone called Jeff Buhrt added the fault tolerant compression feature to afio, and distributed the improved version, probably by posting it to a Usenet newsgroup. In any case, soon afterwards an Englishman called Dave Gymer downloaded afio from Usenet and started using it. In 1993 he made a Linux port, and uploaded it to sunsite.unc.edu, then the major FTP site for Linux application software.

How does one get involved in an open source project?

At one point in early 1993, I had a bad experience with the fault tolerance of tar, so I went looking on the Linux FTP sites for a more fault tolerant program. Afio was the most fault tolerant program I could find. Afio did not have all the types of fault tolerance I wanted, so I started writing my own backup program called tbackup which would use afio as a main component. The main added feature of tbackup was the fault tolerant and user friendly handling of cheap floppies as a backup medium – I had many boxes of cheap floppies lying around, containing outdated versions of the Slackware Linux distribution.

Pretty soon I found that afio would crash in compressing large files if the hard disk was nearly full. So I changed the code to fix that, and also mailed the changes to Dave Gymer, for him to incorporate in new afio releases. Fixing the code and mailing the fixes were fairly natural things for me to do. From my late 1980s home computer hobbyist days I was used to writing my own improvements to other people's code. Also I had already been using BBS systems and Internet mail for some time, so I was used to communicating over the net with complete strangers. It was obvious that the Linux community was just another bunch of computer hobbyists working in a gift economy, similar to my old home computer club. All things considered, it required no big conceptual leap for me to become a Linux open source contributor. It was just a combination of behavioral patterns and rules that I knew already.

At one point, in an e-mail exchange with Dave, we came to the joint conclusion that I was making more frequent changes to the afio code than he was. So we agreed that I should make the future afio releases. In December 1993 I uploaded a new version of afio, version '2.3.5 for Linux', to the usual Linux FTP sites. I had updated the documentation so that future bug reports would be sent to me. I was now the official maintainer of afio for Linux. People with other Unix versions also picked up the new afio for Linux version, ported it to their systems, and sent me portability patches. So after a few versions I dropped the 'for Linux' from the version designation.

What does an open source maintainer do?

Here is the process that I use to maintain afio. It has remained more or less the same over the years, even though, since 1993, the size and composition of the Linux community has changed drastically. I did not get this process from a textbook, nor did I first study other open source efforts to see how they did it. I started doing it this way because it seemed to be the obvious way to do things. (It is actually an interesting question if the open-source community is self-selecting for personality types to whom a certain way of working is the obvious way to work. Reading media accounts of how other people in open source do things, what strikes me most is that I find everything completely obvious, while the journalist writing it often expresses wonder at how things are done.)

Getting e-mail about afio

A big part of the maintenance process is dealing with the e-mail I get about afio. I get on average about 5 new mails related to afio each month. I can answer about half of these mails with a single reply, the rest lead to a series of message exchanges. On average, handling the mail takes me about 10 hours per month. I am very careful in archiving all the mail, to make sure that I will account for all contributions and bug reports when making the next release.

I try to reply to every new message within a week – if I have no time in that week to address the message I just send back a short reply that I am very busy and that I will give a full response hopefully within N weeks. I consciously work to give the impression that something will really happen with the mail people send me. The last thing I want is to discourage people from sending me more contributions in future. Of course nothing bad will happen to me when my correspondents get unhappy about the way I treat their mail. But I am working from the principle that everything worth doing is worth doing well. As long as I choose to fulfill this role as a maintainer, I want to keep up the same standards of service that I would like to see in any other software project, be it commercial or open source.

I use an informal tone in my replies to e-mails, but I consciously try to be polite and clear, even if I think that the original question is stupid or wastes my time. I actually get very few stupid questions, and most of the mail I get comes, as far as I can determine, from people who are already somewhat experienced as a Linux or Unix system administrator. In the last year I have started to get some mail from commercial Linux system support companies, who are asking me about problems reported by their customers. Again I treat these the same as any other mail.

Mail with questions

About half of the e-mail I get is some kind of question: how can I do X with afio? What does this error message mean? Is it available on platform Y? I can usually address these questions with a single reply. Sometimes I can only give a preliminary diagnosis and have to ask for clarifications or more information. In about half of the cases, if I ask for a clarification of a vague question, I do not get any reply. Presumably the person asking solved the problem already. About half of the questions I get are answered in some way by the manual page or release notes. I don't know the complete manual page by hart, so I usually have to look myself to see if the answer is in the documentation – if I find it I summarize what to do and then often cut-and-paste from the manual page in the reply. About half the questions I get uncover some weakness in the documentation, which I then often fix in the next release.

Mail with bug and problem reports

A second class of e-mail, about one fourth of the total volume, reports some afio behavior that was not expected by the user. The message I get can be a detailed bug report, but most often I get a cut-and-paste of the afio command line used, some error messages, and a partial description of the system configuration on which afio was run. Sometimes the observed behavior is actually correct according to the manual, and the user just expected something different. More often the behavior is something that should really not happen. I always end replies to such messages with some variant of 'thank you for this bug report'.

Sometimes I have seen a similar problem reported before, and I can search back in my mail archives and software change log to find a reply. If the problem is new, I try to reproduce it on my own machine. I manage to reproduce it about one third of the time. If I cannot reproduce it I will ask for more information, or for the results of some specific tests. In the end, about one third of the reported problems remains unresolved – sometimes with suspicions that the real source of the problem was a bug in the device driver of the backup device, but often with none of us having a real clue about what went wrong. Leaving something unresolved is frustrating, but at one point I have to decide to stop trying. Often, my correspondents are happy anyway when I tell them that I have given up, because in additional tests they ran the problem never occurred again.

Mail with contributions

A third class of e-mail, about one fifth of the total, contains a contribution to the afio code or documentation. Contributions to the documentation are

fairly rare. I usually get code, in the form of patch files. About one third of the code contributions are bug fixes, about one third compatibility fixes to port afio to a non-Linux platform, and about one third are new features. When I get code for a new feature, it rarely includes any documentation that is good enough to paste directly into the manual page. I never ask contributers to write the missing documentation, I just write it myself. Many of them are not native speakers of English anyway, and I would not want to annoy them by drastically re-writing their attempts before inclusion.

In a few cases I reject code for a new feature, saying that I will not fold it into the release. This is either because I believe that the function can already be achieved in another way, or because I believe that the feature is just a bad idea, that would take too much of my own time to fold in and test. However, overall I hardly ever reject anything, and as a result the number of command line options to afio has grown from 36 in 1993 to 60 now, using all lower case letters, all upper case letters, and most of the numbers as option flags.

I always make sure that I give feedback to code submissions, either with a statement that I won't put the code in, or more usually with a statement that 'I will probably add it to the next release, which will come out in [time estimate in months]'. I write that I will 'probably add', because at the time of replying I have not yet made a full evaluation of the contributed code. I only take a very close look at the code when I start to prepare the next release.

Other mail

I also get a few e-mails which do not fit any of the above categories. Very rarely I get a 'thanks for afio' message without any further questions or requests. Very rarely, I get a plain request for a new feature. Sometimes this feature is something that can already be done with afio: if so then I write back how to do it. If it cannot be done yet, I generally comment on whether and how it could be implemented, and encourage the requester to send in an implementation. Usually I do not get any. Sometimes, if I believe that the idea for the feature is a good one anyway, I implement it myself at the time of the next release.

Making a release

I do not release new afio versions often. In the last few years, the release frequency has been about once every 9 months. A few releases were prompted by the discovery of critical bugs that should be fixed urgently, but usually I release when I have a sufficiently large number of patches and bug reports, and when I can find the time to fold them all in. Making a new release costs me about 40 working hours: I work in evenings and weekends over a period of a few weeks. During that time, I re-visit all archived mail since the previous release, changing the afio code and documenting the changes as I go along. At the end I do regression testing, create a new source archive, and upload it.

Afio is a mature backup program that people rely on. My first order of business is not to introduce any additional bugs, and this drives my release strategy. For other open source programs, which are early in their development lifecycle, the strategy is to release very often, relying on the early adopters to find the bugs in the new code. With afio I also rely on users to find bugs, and this user testing adds significant value, but the type of bugs people find are the very obscure bugs that are left in a well-aged program. For example, a recent bug report had to do with the incorrect handling of Unix file system symbolic links that have several hard links to them. Other bugs that people find in afio are those triggered by new use cases. Over the years, as hardware capacity grew, I first got bug reports related to making multi-tape archives larger than 2 GB (2^{31} bytes), then about handling tapes that are individually larger than 2 GB, then about archiving single files that are larger than 2 GB.

The release process

When making a new release, I carefully hand-check and test all code contributions. Often I make substantial changes to contributed code to make it more safe or general. The new idea behind the code, or the finding of the bug that the code fixes, is often the most valuable part of a code contribution. Reviewing and testing new code takes significant work, but it is needed to maintain the most important feature of afio: its stability. This careful review of all code is not unique to afio: I have also seen it in the maintenance of the Apache HTTP server, another mature

piece of open source software.

Coding is actually a very small part of making a release. I spend most of my time testing and updating the documentation. When I add a new feature, I often spend more time updating the manual page than updating the source code – writing a terse but complete description of a new feature and its limitations is surprisingly hard. I also spend significant time updating the change log file that is bundled with the afio sources – see figure 2 for a typical excerpt.

> *Version 2.4.7:*
>
> *Fixed bug that sometimes caused '– compressed' to be printed twice in verify operation. Has to do with not flushing stdout, stderr before forking. Bug reported by JP Vossen.*
>
> *Added more material on how pattern matching works in the -y option section of the manpage, and added examples of selective restores to manpage. Based on questions by Kjell Palmius and Stojan Rancic.*
>
> *Added text to BUGS section about afio not being able to write into directories for which it has no write permissions, except when running as root. Problem reported by Kagan Kayal.*

Figure 2: A part of the afio change log

The change log has two main functions. First, it helps me and other contributers to keep track of changes and solved problems. Having a very detailed change log saves me significant time in answering e-mails. The second function of the change log is to record the names of all contributers to afio, which I define as everybody who sent me any mail that led to changes in the next release. The change log gives visible evidence that even the smallest contributions are welcome, and will have an effect. I have a theory that this is very important in encouraging future contributions. I never record the e-mail addresses of contributers in the change log, because automatic publication of their e-mail address might actually discourage people from sending me mail.

The last part of the release process is to make a new source archive, upload it to the various repositories, and write an announce message for the comp.os.linux.announce newsgroup. This always takes a surprising amount of time, because you do not want to make any last-minute mistakes. I need to spend some time for every new release to catch up with recent changes in publishing Linux software. Back in 1993 it was sources on FTP sites and a message on an announce newsgroup. Now it is mostly web sites and pre-packaged pre-compiled binaries. However, I still don't build pre-packaged binaries, and I do not get deeply into the web site stuff: I have decided that I'd rather spend my time on other things. Because I don't make pre-compiled binaries, afio probably has less users than it could have. But that is fine with me – I am not in this for world domination, and have no obligation to spend my time serving all Linux users optimally. My main interest is to give to other programmers in the open source community, who will be served about as well with a source release. Somebody else, in the Debian Linux distribution effort, does in fact maintain a Debian binary release of afio, and we have very friendly relations. When I am about to make a new source release, I give him an advance warning. When he gets a bug report about the binary release that has to do with the source, he copies it to me.

What is the motivation for doing it?

Software maintenance is perhaps not a very obvious thing to do in your spare time. My reasons for doing it are partly historical. I started out mainly as an open source author – I like writing software and if I do it in my spare time I like to share it with others. Over the years, I found that I had less and less spare time which I wanted to devote to programming. So I stopped writing new code and only did maintenance on the old code in the projects that I happened to run. By 1995 I was doing maintenance on 3 open source projects: afio, tbackup, and futplex. Still I had less and less time: I found that my release frequency was dropping to below what I found reasonable. So after 1995, I first stopped maintaining futplex, which had never attracted a very active user community anyway. Later still, I also stopped maintaining tbackup, which did have a user community, but one that generated much less feedback than the users of afio. I still get messages about tbackup, about one every two months. I always reply that the software is now 'dead' and unsupported, and that I recommend using another backup program even if tbackup does still install.

Maintaining afio is not very creative work, though there is occasionally an interesting puzzle. Do-

ing the maintenance process is mainly rewarding, I guess, in the same way that gardening is rewarding: there is usually no great pressure, you get to do something immediately visible with your own hands, and it is nice to make things more tidy. I also find it rewarding to help complete strangers who e-mail me. I am a heavy user of open source software myself, and it feels good to be in a position where I am not just taking, but also giving back to the open source community. Occasionally, it is fun to consider that I have achieved some degree of immortality through my work, because the code I wrote has been pressed on lots of CD-ROMs. However this is more of a fringe benefit, it is not a state of mind that I can sustain indefinitely.

Starting an open source project

Starting an open source project is easy. Create some software (but make sure that you do not incorporate any third party code that is restricted or proprietary), bundle the sources together with a file that identifies you as the author/maintaner, and upload the result to whatever the usual places are for your platform. That is really all there is to it. A web page is optional, but very much expected these days.

Of course there is no guarantee that a new project will generate the level of interest that you expect. In my own new projects, I never tried to predict or optimize the level of interest beforehand. When my code was at a stage where I thought it to be potentially useful to others, I just wrote the amount of documentation that I judged to be necessary to support other users and developers, bundled it, and made the first public release.

Though I did not try to guess levels of interest beforehand, the levels that I observe during a project do influence my actions, in particular when choosing which projects to stop.

Further reading

In recent years, the new economy has become somewhat of a spectator sport, generating a matching volume of mass media accounts and Internet discussions. Microsoft litigation and open source are particularly popular subtopics. On the Internet, participating in discussions about open source has become a legitimate full-time hobby all by itself. Most of the written material can be safely ignored, unless you happen to be a fan.

The Cathedral and the Bazaar by Eric Raymond [1] is a classic and thought-provoking essay that contains some good descriptions of the open source process. The essay is thought-provoking because it argues, with some actual proof, that very complex software projects can be run successfully, even optimally, without much central planning. I don't believe all claims of the essay, but that is exactly what makes it thought-provoking. Don't miss the [EGCS] end note which is present in the more recent releases of this essay. There are several followup essays by the same author, but I find these less thought-provoking.

Hackers by Steven Levy [2] is a well-written book, written in 1984, about several open source cultures that pre-date Linux, about their development processes, and about their interactions with the market economy.

A recent statistical survey of open source authors is at [3]. Like many statistical surveys, it offers few real surprises, but it does solidly contradict the view that open source authors are an untrained mob of young computer nerds out to destroy Microsoft.

References

[1] http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/

[2] Hackers, by Steven Levy. 1984. http://mosaic.echonyc.com/~steven/hackers.html

[3] http://www.osdn.com/bcg/

Koen Holtman is a researcher at the California Institute of Technology, where he does architecture and coordination work in developing the world-wide distributed data processing infrastructure of the CMS particle physics experiment. The current version of this infrastructure relies heavily on Linux systems, and several of the development partners publish their middleware with an open source license.

Koen was an OOTI from 1995-2000, finishing with a Ph.D. in software design for work performed at CERN in Geneva, Switzerland. He also holds an engineering degree in computing science from the Eindhoven University of Technology. He has been active in the Linux open source community since 1993, and also contributed code to the Apache server project.