Seven reasons I probably can't help you get my open source software running on your computer

Posted on April 13, 2015

The EL:DIABLO/PETRARCH system is beginning to get some traction, and with this, we are starting to get increasing numbers of emails asking for assistance. In a few instances, these can be resolved and in a very, very few instances, they have alerted us to important bugs. However—there's probably a rule of thumb here—if an issue can't be resolved in two or three email exchanges, it usually cannot be resolved at all.

This, I am sure, ends up being very frustrating to the person who originally sought the assistance, and it almost certainly leads to some variant on the following generalization about open source in general and programmers in particular: Open source is too good to be true, and programmers are a bunch of introverted overpaid arrogant bastards with inadequate personal hygiene who deliberately obfuscate their work to make it seem more difficult than it really is even as they pretend to provide it to you for free. Working on a script?: Have a programmer eviscerated by a velociraptor, crushed in a trash compactor, or reduced to Valyrian dragon chow. [1] All of these are sure crowd-pleasers.

Well, it's actually a little more complicated than that, and hence I am devoting a bit of time—also provided for free—to explaining why.

1. No two research computers are identical

For purposes of this discussion, we will stipulate that the software you are trying to use works when correctly installed. In the case of EL:DI/PETR, we've been running the system 24/7 for about six months and have encountered a single untrapped bug (which we corrected, or rather, trapped). The major open source projects—R, LaTeX, perl, Python, NumPy—have accumulated millions of hours of use on tens of thousands of installations and thus, you can reasonably assume, also work.[2]

So why won't it work on your computer?

In the absence of access to your machine—and please, don't ever, ever grant access to your machine to someone you just know via the internet—the basic answer is "we haven't a clue." In situations where one is experimenting with open source software—as opposed, for example, to a standardized telemarketing operation selling the rights, smirk, smirk, to gold bars to protect you

from the massive hyperinflation of 2009-2011 that destroyed the value of the US dollar[3]—it is safe to assume that on the software side, no two computers are identical. I don't know what else you have on your machine, or how it is installed: In a complex system like Python, or even EL:DI/PETR, there are multiple ways of doing this, and they all work, but they work differently. I also don't know your directory structures, and probably nine times out of ten, that's where the problem is.

It is even worse when you are in an institutional setting since your machine may be deliberately configured, for good reasons or bad, so that it is *impossible* to run the software. I've known institutional IT specialists I would trust to do almost anything correctly, and I've known some who would have difficulty operating a vending machine.[3] Again, I have essentially no information on how your machine is configured, nor any ethical way of obtaining that information. This is a problem.

A Story: Last summer I spent the better part of two weeks trying, via email, to help a graduate student in Europe get TABARI running on a Linux installation. I've run TABARI on dozens of Linux systems, nothing new there, but hadn't seen this issue, and could not debug it remotely: The effort was unsuccessful.

A few weeks later I upgraded my software to the latest version of the GNU compiler suite, and suddenly TABARI would not compile for me either. In fifteen minutes, I'd traced the issue to a simple problem in the 'make' file, and moving a single file name from the beginning of a list to the end solved everything. I'm sure changing the 'make' utility made sense to someone somewhere, and probably even makes the whole GNU suite more consistent, but it broke this particular piece of code. The person I was trying to help happened to be just a bit ahead of the curve in getting an up-to-date installation.

Email exchanges, by the way, are particularly inefficient because of the lag time: I solved the problem in fifteen minutes because I could run however many experiments—dozens, typically—each taking a few seconds (try this, didn't work; try that, didn't work, rinse and repeat). Email: we're talking weeks.

2. You shouldn't be using open source software unless you accept the core elements of the social contract of open source.

Open source software is only "free as in puppy." Entire books have been written about why open source works, and let me say that for most programmers, particularly those working

independently who have to provide their own tools, it has been an utter and complete game changer, about as far from "too good to be true" as you can imagine. Others have written more extensively on the implicit rules of open source software, but for our purposes I'd emphasize:

1. We have provided you with every single line of code so there is no possible issue with the software that you cannot, given sufficient knowledge and effort, figure out.
2. In addition to whatever intrinsic rewards we may obtain [4], we are posting this code because we want it to be improved: steel sharpens steel.
3. As with every complex social system involving coordination for a common good [6] working effectively in the open source world involves internalizing and following a set of cultural norms.

Or rather "sets", as there are multiple cultures, for example those of Python, javascript and R. The most obvious difference is in the skill set the user is presumed to have already mastered but there are others: for example I think generally web-based programming forums such as those for javascript and CSS are noticeably more polite and social than Linux or C forums.

A Story: Ever tried playing in a bluegrass, Irish or old-time "jam", those informal gatherings of musicians of multiple skill levels? These typically start out really easy, with slow, familiar tunes, lots of open chords, and pretty much everyone can contribute. But then a point comes when the banjo player is channeling Earl Scruggs and the mandolin player Bill Monroe and the fiddler must, indeed, have made a deal directly with Satan (Satanic fiddlers, I repeat myself). At that point, unless you have had a heck of a lot of practice, there is no way you can possibly keep up and the best thing to do is just put down your instrument and enjoy the show.[7]

A really good musician can work in multiple genres, but a medium level bluegrass player is not going to immediately become a medium-level Irish player, or vice-versa. The fact that you are really skilled in R doesn't mean you've got a comparable level of skill in Python, or vice versa, though you are probably well on the way to getting there. But there is a still a learning curve and at first you are going to get hung up on some really simple things.

Open source is like that. There are levels where everyone can play: certainly you should be able to do "Hello, World!" in pretty much any language, and then as you work your way up things get harder—say to the Python statistical data management system Pandas, which appears to be channeling R, and quite possibly Satan—at some point the software will get beyond your skill level. For example despite almost 50 years of programming experience [8] , I would not even begin to think I could touch the Linux kernel code. EL:DI/PETR, with three or four complex parts, and multiple authors, is heading towards the upper range: it isn't the Linux kernel, but

getting it fully deployed is a lot complicated than deploying its predecessor, TABARI. [9]

3. Wizardry is hard.

One of the many things I like about J.K. Rowling's *Harry Potter* series is that learning to be a wizard is hard work: potions can take days to concoct, and still fail, and there are spells that take years to master and some people never really figure them out.

Yep, sounds a lot like programming to me, which is probably why most programmers really like *Harry Potter*.

Not everything is hard, and actually, you never know for sure just what will be hard. Sometimes things work right away; sometimes they can take hours to get it working even when the final correction is just a couple lines—or characters—of code. That was certainly my experience with installing one of the major Python packages—I believe it was NumPy—which after several hours I finally traced to having incorrectly installed something earlier. If you are willing to put in that sort of work, open source provides an astonishing set of opportunities. If you aren't, at some point it isn't going to work for you.

A Story: You've all heard the ca. 1955 three rules for making it through life, right?

- never eat at a place named Mom's
- never play poker with a man named Doc
- never get into bed with someone crazier than you are [10]

Here's the analogue for finding a good programmer:

- Programming is 10% book learning and 90% experience
- The programmers who you might hire will vary by at least a factor of ten in terms of the amount of time it will take them to complete a task; the very best could have an edge of one-hundred, and a remarkable number of people who say they can do the job will not be able to finish it at all
- Like competent craftspeople in every place and age, they are in short supply: there are never enough good programmers for the tasks requiring a good programmer [11]

This is not to discourage people from learning programming, starting with the 50% of the population who were commonly programmers when I first learned FORTRAN [12] but currently are discouraged from entering the field. But it will take quite an investment of time, particularly

with the complexity of current environments, and, in my experience, at least some aptitude for the task.[13] However, the resources now available are vast and very accessible and in the end you will find that most skillful programmers are not merely introverted arrogant bastards best suited for dragon chow. A few aren't even introverted! The pay and independence isn't bad either.

4. Critical thinking is great, but problem solving skills are even better.

Some instructional programs do this really well, and others do not. More generally, to the massive frustration of those who will not rest until they've increased income inequality to the level of the late Roman Empire [14] programming remains a craft, not a simple set of routine tasks: You learn a set of tools but then it can take years to figure out how to best apply them.

As for assuming that learning programming can't be all that hard and anyone can do it—and for some tasks, it isn't that hard, and if that's your situation, ignore the following—let's go back to the musical analogy:[15] You're putting together a bluegrass band, and you need a banjo player. Which strategy do you think is going to have a better outcome?: call your worthless nephew Bob and say "Bob, stop smoking that joint and go learn to play the banjo." Or find someone who already knows how to play the banjo—ideally, bluegrass banjo—who you can persuade to join the band.

A Story: A while back I was involved on the edges of a small construction project with a guy who has been doing carpentry for about as long as I've been doing programming. There were no specific plans, just a building site, a load of lumber, and—definitely—a nail gun. As I watched the guy at work, I realized that the difference between how he did the project and how I would have done the same thing is that he was always thinking three or four steps ahead, and, out of long experience, doing things that would prevent problems further down the line. That's a pretty good definition of an expert, and much of the reason an expert can do things faster and more effectively than someone just watching Home Depot videos on YouTube. In a few hours that pile of lumber was a woodshed, and I suspect it will be standing longer than I will.

5. Software is neither static nor, particularly across systems, logically consistent.

Software, particularly open source, is organic and evolves, but for that same reason, it exists in a

series of punctuated equilibria. Once something works it will probably be maintained, until it isn't. To function in this environment, you need the twenty-first century equivalent of horse sense, which works because while half the software one is using probably didn't exist five years ago, other parts existed forty years ago.

A Story. About three weeks ago we noticed that the big black plastic wireless device in our guest bedroom that exists primarily to consume extortionately-priced ink cartridges was now simply an inert big black plastic brick providing the final resting place for stink bugs. We had changed nothing: it just stopped responding. Per the haiku that I believe was the motto of the Windows development team during the "Blue Screen of Death" decades:

Your computer was fine
But now it will do nothing
Sad, so very sad

I went onto the web and found we were definitely not the only people on the planet who were unexpectedly having this experience, and tried various of the suggested solutions to no avail. In the end, I simply unplugged both the printer and our wireless router, plugged them in again, and everything worked.

Computers are like that.

6. Simple is really hard, and really expensive. So is documentation.

Though if the software comes with documentation, you should at least read it. Yes, yes, we know that you didn't need to read the documentation when you bought a system from one of those companies that has more money in their [off-shore, tax-sheltered] bank account than is held by the US Treasury, but a lot of open-source software was not written by such companies.[16] And virtually all open source software was written to solve sets of problems which may well not be precisely the same set of problems you are trying to solve.

To take our proximate case, EL:DI/PETR is already at a level of complexity where I don't know the purpose of every line of code. We still don't have a large developer community, but the system involves very substantial development by not only me, but by John Beieler.[17] Do I understand all of Beieler's code?: no, I just know that it works. Neither Beieler nor I even begin to understand all of the code in Stanford's CoreNLP suite. Which also works. Usually.

I actually did understand every line of code in TABARI, but that was a ca. 2000 program with

only limited open source involvement. And even with TABARI, there is a rule of thumb that any code you wrote more than six months ago might as well have been written by another person. It takes a couple of hours for me to get back into the code in any significant part of PETR, the last time I was doing serious programming on it was four months ago, and in the intervening period I've worked on three or four other projects of comparable complexity.

A Story: I will not provide a story here: go read the documentation. Including the comments in the code. And if you find something wrong with the documentation, at least tell us, but better yet, fix it.

7. I'm not working for you

You are probably being rewarded—whether with hourly pay, a stipend, class credit, the prospect of an M.A. or Ph.D. thesis, or whatever—for getting the open source software to run. I am not, and I've already gone the extra step of making the code available with at least some level of documentation.

So, I know what you are thinking: it's that old open-source bait-and-switch! I'm only*pretending* to provide the software but in fact in order to actually use it, you have to pay me. Arrogant bastard programmers! Dragon chow! Dragon chow!

Again, it's a little more complicated than that.

It is true that, as with every contract programmer—and I'd extend this to the many small development groups with only a single level of management, though not to those working for corporations with bank accounts, or VC funding, exceeding the currency reserves of most nation-states—I only get to eat what I kill. So to speak. And contrary to the popular mythology about open source, most is actually written by professional programmers.[18]

But that doesn't mean I want to work for you. For starters, wrapping this entire essay back to Point #1 [19], I probably can't help you unless I can put my hands on your machine.[20] Which is not terribly practical if you are on the other side of the country. Or the planet.

Once I've got that access, assuming we've eliminated the problems that could be resolved by reading the documentation, getting the software to work would probably take me anywhere from fifteen minutes to a couple hours, but, returning again to Point #1, I have absolutely no way of predicting what amount of time will be involved. For that amount of work, it still makes little sense for you to hire me.[21]

In the meantime, the time I'm spending doing what is really *your* job is taking away from the time I have available for projects where I have an on-going relationship with the client and I'd really like to deliver them a quality product more or less when I said I would. Responding to a probably impossible email request does not contribute to that end.

So yes, if you can't figure out the software, you probably need to hire a programmer, but a local programmer, and someone with administrative access to your equipment, and probably someone you will be asking to do work on more than one occasion. Because they are unfamiliar with EL:DI/PETR, they probably won't be able to solve the problem in fifteen minutes, though they probably will be able to solve it. If no one with that skill set exists in your vicinity, I'd say you've just found a business opportunity. But not one that I can solve. I have promises to keep and, indeed, miles to go before I sleep.

A Story: My wife took on the task of getting our ComCast account moved from State College to Charlottesville, a process involving roughly the same degree of complexity and effort as the Iranian nuclear program negotiations. After one particularly extended—if futile—conversation with a ComCast rep, she got him to say where he was located. "I'm in the Philippines. The rainy season is coming, and my bosses are all really crabby." We eventually got the system working, then squirrels ate through the coaxial cable.[22]

Footnotes

1. This occurs in Season Six: George R. R. Martin is going kinda slow writing the books so the script writers are going to need to get kinda creative.

2. One of the reasons I've switched to working almost entirely with open source software is I was finding more bugs in the propreitary software I was working with than in the open source.

3. You remember the dramatic dollar hyper-inflation of 2009-2011, right? When liberals who had foolishly ignored the copious warnings on Fox and conservative talk radio were reduced to using rolled-up Obama posters to kill rats for food?

4. What is printed on the bottom of cans of RedBull sold at Microsoft [substitute Apple or Google per your preference]? "Open other end."

Though this is nothing compared to the IT people employed at the lower levels in academia,

where salaries are typically not competitive with the private sector and, as the saying goes, they usually don't get the sharpest crayons in box. In university research centers, this is less of a problem, since the attractiveness of working on a variety of problems and [sometimes] with state-of-the-art equipment and software compensates. But if the job is routine and doesn't pay competitively, you're probably looking at a pretty much continuous mess'o'trouble.

5. The ego boosts are real: I still remember the first time, this when open source was still fairly new, when I saw that a rather esoteric program—it implemented the algorithm for fitting hidden Markov models—that I'd converted from someone else's open source code in C to Pascal (or maybe it was the other way around) was being used by some electrical engineers in a publication. And about the same time, some undergraduate at Stanford cleaned up the TABARI code so it no longer generated warnings when compiled on Linux.

6. Which is to say, every human system except those postulated by "rational choice" economists and political scientists, who like *Alice in Wonderland's* Red Queen can only thrive in their tenured sinecures by learning to believe six impossible things before breakfast. I digress. Though it isn't just six.

7. In the EL:DI/PETR world: just use the data.

8. I exaggerate: it's really only been 48 years.

9. The *code* in TABARI, written in C/C++, is substantially more complicated, which is also why TABARI is about fifteen times faster than PETRARCH, but the full system in EL:DI/PETR is more complex. And of course does a lot more.

10. There's a fourth: if you can't identify the sucker at a poker table, it's you.

11. Consequently, if someone with an active research program suggests you hire one of their students…

12. "Ada Lovelace wrote the first loop. I will never forget that. None of us ever will." Admiral Grace Hopper.

13. Luthier Wayne Henderson, channelling a Victorian-era joke, says that all you need to do to make a guitar is take a bunch of wood and a pocket knife and carve away everything that doesn't look like a guitar. There are times I feel like something along those lines can happen, in reverse, with a program.

That is, sometimes I'll sketch a design, but particularly on smaller projects where I'm unlikely to re-use the program, I'll just sit down and start writing code, not necessarily consciously knowing why, but confident that at some point I'm going to need that code because I've written similar programs many times in the past. So I suppose, following Henderson, a lot of what I do is just take an empty text file and keep adding chunks of code until it works like a program.

But don't try this at home.

14. Actually, it's already at that level, but that's for another discussion.

15. Yes, I see another blog entry emerging here…

16. Though quite a bit is.

17. On GitHub, you can see exactly who contributed what.

18. Who is a professional programmer? If that's in your job description, it's an easy determination. But if, like me, you've done contract work on a variety of tasks—my degrees are in mathematics and political science, not computer science, which wasn't even a major when I was an undergraduate—it comes along more gradually. In such a situation, I'd say being a "professional" means is getting hired, on multiple occasions, to write software someone else is going to use for their work. Quite a few years ago I started adding up how much I'd been paid over time for writing such software and realized it came to multiple hundreds of thousands of dollars, at which point I figured I was a "professional."

Once again, there is a good analogy with musicians. You're an orchestra or studio musician with a union card: sure. But you start out playing for fun, then for beer money, and then at weddings, then you record a CD and it sells: when does the transition occur?

19. Or, in musical theory, resolving the essay to the opening key. A pretty good technique in blogs, it turns out.

20. Not in the sense of Oral Roberts, though at times that appears to help as well.

21. Which is to recall a particularly notorious tax form that Pennsylvania requires of LLCs which is so bad—it runs some ten pages, and I swear probably contains a provision absolving any corporation which provides free natural gas to kitchen faucets in northwest Pennsylvania of all tax liabilities—that accountants refuse to do it. I asked. I did my best, and concluded I owed

about $250 only to receive a notice some eight months later that I'd done the calculations incorrectly—certainly no surprise there, as the 38 pages of instructions appear to be a bad translation from proto-Hittite [23]—and actually owed only $21. No refund check, however: that's probably a twenty-page form. Pennsylvania: the state where the only task the government performs competently is incentivizing citizens to purchase wine in New Jersey, Maryland and Ohio.

22. This story, you will notice, has nothing to do with the rest of the entry, but mouseCorp—slave-drivers—pays me by the word. Blogs are like that.

23. There really are 38 pages of instructions. But they weren't translated from proto-Hittite but rather they written in something far worse: fluently crafted lobbyistian. Which is to say, most of that form exists to provide massive tax breaks to a tiny number of individuals, but this is quite deliberately written in a fashion that makes it nearly impossible to figure out who that beneficiary is. And as part of the game, the beneficiary will, of course, be constantly complaining about how unfair the tax system is.

And, well, it's pretty unlikely that going to change: here's a nice article on why you are never going to see the IRS provide any user-friendly software and—I'm shocked, shocked—lobbying money is involved. And it co-occurs with an article on the privatization of security—that old Weberian "sovereign monopoly on the legitimate use of force" was *sooo* twentieth-century—and presumably in due order we will see the slaughtering of unarmed black men out-sourced, with the help of Erik Prince, to Chinese "security" guards operating with impunity. And not just in Africa. Suggesting, once again, that we've moved into a system that looks a heck of a lot more like Rome under the Borgias than Wisconsin under the La Follettes. A topic for a later set of entries.

Posted in Uncategorized | Leave a comment | Edit