Church of Emacs 2.0

A sort of Vatican 2.0 for the Church of Emacs

Third Draft Version, Sept. 2018

Preface: Short Books

I've written a few short books. I wrote an extensive tutorial about basic programming with 20 short chapters, which years later turned into a new programming language/dialect and a second book (and one or two other dialects as well.)

I've written a short book about the Free Media Alliance, another about holographic learning-- and obviously I've written this one as well.

These are not intended to be incredible feats. If I wanted to spend a lot more time, have something more professional, or impress harder-to-impress people, I could spend a lot more time writing longer books. But that's not why I write them. These books start out as rambling posts online, which reach a point where I say "Alright, obviously you're not going to shut up about this until it's a little book."

Some may even dispute that it's a book. Call it whatever you want to, there's a place that still sells "zines" for $10 each that use 10 to 20 pieces of paper stapled together. I go for that form factor (I have yet to actually paginate for that-- ebooks are so easy these days) though some of these titles would take more pages. I'm just going to call this a "short book" for now.

Everything in it is in the public domain, so if part of it is useful to you-- grab that part, and copy it somewhere. Thank you.

Chapter 1: Why Even Bother?

The GNU operating system works like a charm. It is being used to write this book. I stopped using Windows on any of my computers more than 10 years ago. Since this is my first book to the free software community in particular, many of you went Windows-free and Apple-free ages before I did.

I don't compile my own operating system. I used to run a server. I used DOS and Windows 3.0, and I know what a truly unstable operating system is like to use, (though I've never experienced the joy of line editing over bad phone lines with a teletype.)

I have on more than one occasion used binaries from foreign distros, as if I am not dooming the potential uptime of my desktop to something less than 2 years. I used to be one of the people who (rightfully) frown at this. But I do it anyway.

There are many reasons to promote free software, other than purely for free software's sake. The internet is the largest library in the history of mankind, at a time when companies like Amazon pose an existential threat to libraries. The increasingly digital, increasingly fettered computing that Richard Stallman predicted lends greater credence to his warnings all the time. He is fundamentally correct: without free software, our society is shackled by its own 1's and 0's.

Normally, this is where a critique of the FSF, free software or Stallman himself would turn from the nod to his legacy to all the reasons he's no longer relevant. We have open source, Microsoft loves "Linux" (even if they spelled "GNU" wrong) and Firefox (sorry, IceCat) is good enough.

In the endless debate about why you should replace the name "GNU" with the name of a kernel, a very simple solution is offered-- share credit with a brand used increasingly to attack and undermine our digital freedom. Yes, Linux is a fantastic kernel. Even though the BSD kernel has some major, impressive advantages, I favour hardware support and software support, so Linux is my favourite kernel in the world.

Open source tells its story from its own perspective-- by their criteria, we have already "lost" and should just give up and cede to the Linux brand. But the truth is, we don't measure success (not primarily) in marketshare. It would be silly.

Our success comes from setting goals that move us closer to freedom, and meeting those goals. Thus while we succeed year after year in meeting goals, open source paints that as a failure because we aren't as successful at the goals *they* consider more important. This is sort of like saying that Michael Jordan is a failed athlete, because he sucks at baseball.

Quite understandably, the Free Software Foundation is structured around skilled coders, a few large sponsors, a very large number of smaller sponsors, and the GNU (free software) ecosystem. Open source is structured around corporate monopolies, and organisations that want to be cozier with them. For a while, Bruce Perens thought this was a good way to promote Free Software. Not long after he co-founded the Open Source Initiative, he resigned with an open letter entitled "It's time to talk about Free Software again."

The goal of open source is to eclipse free software. It has tried, for many many years, to reframe free software as the older, clumsier, boring version of itself. But only the free software movement can ultimately set the goals of the free software movement. This book will set exactly zero goals for free software. It will offer ideas you can use, and outline several things that "we are already doing."

Perhaps those ideas can be done more, or something. But the ideas in this book are all suggestions.

With the increasing number of problems that free software is applicable to as a solution, the first suggestion this book will make is that free software find a way to "build out" from the core of what it already does. The words I put forward in my latest internet rambling were something along the lines of:

"Find a way to double the FSF's success, with less than twice as much funding."

Easy enough to say, right? But this book will explore ideas about how to build out, and how to possibly double the success of the FSF.

Hopefully it will also show that more is at stake than getting closer to a world where all software is free as in freedom.

Chapter 2: Trolls Rule The Earth

If you want to change the world, you have to understand it.

I like to think there are two kinds of trolls-- the rarely-encountered good troll, like a cute harmless prank (or clever art installation) played on you by a true friend or peer; and on the other hand the better-known, evil awful person, who tries to suck the soul out of you one jerk-move at a time.

I'm pretty sure most people still think of narcissism as just an inflated sense of self. That definition may have validity but is not too good, when every idealist is trying to find some way to save the world. Oh, you don't want to use software that doesn't include source code? Boom, you're a narcissist. You're vegan because you hate plants and want them to die? Narcissist, obviously.

The kind of narcissist I'm referring to instead, is the sort who:

1. pretends to care about you or other people

2. misquotes you and speaks for you and gaslights you

3. uses smear tactics and tries to intimidate you, even as a response for anything they dislike about you at all

4. constantly accuses you of things they are doing themselves-- then says they were just kidding, lighten up

5. plays a hero, pretends to care, but whose actions never match their words

6. plays people and groups against each other, often over incredibly insignificant faults

7. has consistently different standards for what they will tolerate vs. what they will dump on you

Narcissists do not respond (initially, later on, after repeated attempts, or under any circumstances whatsoever) to logic or honesty with logic or honesty. They only ever double down with fallacy and lies. Although people say "don't feed the troll," what they don't tell you is that the thing you're feeding them is your happiness and well-being.

This is not just about trolls-- Narcissism explains most of the ills that society has. People think that narcissism is rare, but it is not as rare as many assume and we are creating more of them with a society that is perfect for narcissists. Believe it or not, I don't think selfies are so bad. Prior to camera-phones, they were known as self-portraits.

The real problem with Narcissism is just how many people out there are lying by default, how good they are at lying, and how great they are at weaseling out of any effort to pin them for it. You aren't just wasting your time going after narcissists-- you're wasting your life.

When feminists talk about "Patriarchy" they are describing male narcissism and narcissistic success. I think the reason that males happen to dominate society (at least historically) is that males dominated society (at least historically.)

Historically, the combination of male narcissists, male-dominated politics, and male armed forces meant that men dominated society by default. I personally reject the notion that male narcissism is somehow a "naturally male" or gendered trait-- any gender domination in society is a cultural habit reinforced by differences in physical strength. It's not because "men are just like that."

But narcissists of all genders are "just like that." I've encountered vicious female narcissists, I even nearly married one.

Though they may not always appear to act in groups, narcissists do swarm together. If there's one nearby that you can discern, there are often others lurking around. They feed off your emotions and off the imaginary things they attribute to your feelings-- whether good or bad.

But narcissism helps explain a lot of things-- from non-profits that care more about a fancy, decked-out top office floor than the cause in their mission statement, to the cloying but empty promises in any major political party, to one-sided friendships that seem to always go nowhere (or go crazy) no matter how you work to nurture them from your side, to arguments that start out frustrating and become surreal over time and iteration.

The only protection from trolls is to starve them, and trolls are constantly trying to make good people look like trolls. No matter how many anti-bullying campaigns you run, how many people you ban, how many misguided zero-tolerance policies you write, trolls will thrive if there's food around.

Until the day when everyone educates themselves better about clinical narcissism-- and stops trying to win the argument that a particular troll has it.

If you take down everyone who displays one or two narcissistic traits, you will also stop their victims. You want three things for a victim of narcissistic abuse: You want to give them an opportunity to heal, You want to give them room to speak that the narcissist tried to troll them out of-- and you definitely, definitely want them to fully understand why it is self-destructive to try to go after the troll either directly or publicly.

Turnabout is not fair play-- not just because of karma or some perfect morality-- but because chasing after the troll is just another opportunity for the victim to be abused further.

Many people think this is just about protecting emotionally fragile people's feelings-- or creating a "perfect" code of conduct, or that this is just an opportunity to squash more free speech.

Unfortunately, it can be all those things. And that's a very substantial reason why a global understanding of narcissism would result in a better world, better environments and communication online and offline, **less** perceived need for zero-tolerance policy and censorship and controlled speech, and greater harmony and success.

If you critically examine the news and advertising, we are constantly being played against each other as a society. Corporations do this because it makes us "better consumers" by their definition of "better." (Malleable.) So don't think for a moment that trolls are just some obnoxious kids on an internet forum. Trolls create and sustain monopolies, they use marketing to psychologically manipulate the public, and they create a society in which we cannot work together to do anything meaningful against them.

Understand that power, and you can learn to feed it less.

Will using a fully free operating system help? By no means was all of this said just to sum it up as "use GNU," but yes-- a free operating system would help substantially and in ways that are harder to explain if you aren't already using one.

Using free software, unlike using "open source" is a political and ethical act. Using free software promotes freedom (and choice as well) and it teaches that sometimes, "the shiny" is actually just poison.

That said, there is a lot more to freedom than just software. So many things run on digital platforms now, that the relevance of free software to other (more conventionally though of) freedoms is understated. This chapter is not just about free software-- it is about free society and a better mankind.

Above all, it is most certainly **not** a call for more censorship--but instead, an idea that may help people realise why more censorship is not needed (and wouldn't help much anyway.)

Narcissism is not just male or female, left or right, rich or poor, eastern or western. It is a fundamental evil that has plagued humanity for millennia. But between overpopulation, extremely scientific marketing and global communication, it is very likely that the problem is worse than ever in history.

Chapter 3: GNU Who?

What if I told you that GNU is already one of the strongest brands there is? I mean let's do like John Lennon for a minute, and compare it to a cross. If someone is wearing a cross, odds are, they're a Christian. They could also be Celtic, or in a rock band. Or perhaps hipster irony reaches a new plateau and they start wearing them, too.

You often can't tell what denomination/sect or beliefs a person has just from a cross, but if someone calls their operating system "GNU/Linux"-- that already says a number of a specific things with very good likelihood.

While this is a slightly tongue-in-cheek exaggeration, it's not entirely untrue. Brands start with an image, create some language around it, and then hitch every concept they want you to associate with it to that brand.

The vast majority of the time, such a brand is merely an image-- a face on something that may not be very honest. You learn to associate deeds with that brand, and when those deeds are well known and go against your core values, the owners of the brand try to untarnish it and make it look good again.

The way the FSF maintains their brand is to try to live up to what it stands for, and to do what they claim. I defy you to think of five brands that do a better job than that. So I do think GNU is a very strong brand, though with that said-- if I put the GNU head on a laptop I still expect some people to think it has something to do with a rock band. I used to think Tux was cute too, but after my experience for the past few years he's not so cute anymore. I don't need Tux in my life.

Before Vatican 2, greater (more literal) emphasis was put on John 14:6. Now the church teaches that it won't only be Catholics (or Christians) who find the path. Open source claimed that it would add another path or paths to freedom, though it really hasn't done that. What it has done, is paved the way for Microsoft and Apple to further co-opt free software.

Stallman however, acknowledges that freedom is (obviously) about more than what he and the FSF are working on. There are other paths to freedom, even if free software is an increasingly vital component of freedom around the world.

So where we start first-- and what we get from starting there, depends on who we are and who we "want" to join us. I say "us" in a deliberately vague fashion. I am a free software advocate. I advocate that we use more (and more and increasingly more) free software. I recommend that we refer to it as "free software" and I explain that saying "GNU" lets people know where your priorities are.

People who say "GNU" are far, far less likely to let you down. It's not magic-- it's just a weird brand that people who co-opt free software don't like to be associated with. We are lucky, because they (mostly) make it that easy to tell who's who.

Does calling it "GNU" mean you advocate free software? If you advocate using the GNU operating system, it's a great start. I use it all the time, it is honestly the best system I've ever used, and for the most part it keeps getting better. We have about 3+ years of setback now, thanks to redix. We continue to provide alternatives to redix-- more than I knew, and I try to keep track.

What is redix? Redix is every software-based threat or setback to free software that the FSF does not recognise. It is especially software whose fans tell you that "you will have no choice" in using it-- and who try to stop you from removing it. The unnecessary "cashew" in KDE 4 is perhaps the mildest possible form of redix, parts of GNOME are redix, and practically everything freedesktop.org does is redix.

Redix is not posix, it is not a posix alternative-- it is a parasitic posix replacement that guts posix from the inside and replaces it with a freely-licensed monopoly.

"Ridiculous," the FSF could say. If it is free software, it cannot be a monopoly. We all know that doesn't apply to obfuscated source code. We know that doesn't apply to compiled binaries without source. We should know that it is possible (and undesirable and a problem) to create freely-licensed code designed so that:

1. it is not practical for the majority of distros (including FSF-approved ones) to isolate its components.

2. it has many parts, many of which you do not need or want.

3. it is not designed in a way that a community can practically maintain it-- it can only be practically maintained by a monopoly with paid volunteers.

And above all-- it doesn't only fail to go forwards in these regards, but it actually takes existing components and goes backwards.

Systemd is just one example of redix, it is the redix flagship. It puts too much of your computer at the mercy of systemd developers; all arguments that sidestep this instead of addressing it are beside the point.

The license is beside the point. The number of files the source code is placed in is beside the point. Systemd means less freedom-- and the authors snidely explain that not only do GNU/Linux users not have a choice, but BSD authors do not either.

Again, with narcissists the words and the actions will never match up. Lennart lies about Systemd. No matter what logic you throw at the community that supports it, they will lie and reframe the argument for years on end. After all that, they will blame "neckbeards" who "dont want change."

Until you have dealt with and learned enough about narcissists (a weeks research should be plenty for this purpose) then you may not ever understand how or why the people pushing redix are so diabolical. But such people are common in politics, common online, too common in free software, and extremely (unbelievably) common in open source.

Redix is not a generous gift for you to be thankful for. It is a well-designed effort to take control from the community, and give it away to billion-dollar corporations who like monopoly and control. And if that's not a threat, then we need something additional that's very much like "free software." But it would likely suffice, if the FSF would not neglect this matter for one more year (2014 to 2015, they should have said something better about it) and take this seriously.

Let's be clear about this-- the FSF is not obligated to do anything about it. The only reason they "should" do anything about it is if they don't want to cede more control to monopolies-- which is somewhat of a given. So saying they are "obligated" is entirely wrong. It would simply be a really, really good idea to stand up to this.

"Systemd is free software" really misses the point this time. Yes, you can design free software that is truly bad for free software. Yes, you are "free" to do that. No, it should not be supported by the FSF. Not when the authors brag about how you don't have a choice.

That sort of gloating should be a huge red flag to any free software advocate that something is amiss-- You've heard it from GNOME fans, you've heard it from Systemd authors, and you shouldn't tolerate a coup to threaten the users freedom, no matter how it is licensed.

If someone threatened to put a vulnerability in the kernel-- and then they submitted a kernel (source) patch, and they claimed the patch contained no vulnerabilities-- but it was terribly written, had many functions you didn't want or need, and made it much more difficult to maintain or configure the kernel as you had previously, setting things back for years--

Would you then accept the "neckbeards that don't want to change" argument after all that? From people who continue to insult your intelligence and mock your philosophy of freedom? So why would you ever accept Systemd?

Still, it is extremely important to understand that Systemd really is just one example. Because this is exactly how monopolies will continue to undermine and reduce the viability of free software-- gradually.

After Systemd, Github users were scattered. Did every coder 5who left Github have their project (which they may not have owned, but only participated in) folllow them out? Purchasing Github was like kicking a very large anthill-- yes, there are other places to go.

No, the percentage of "deaths" was perhaps low. But the colony will not get back the time they lost, the damage was done before the merger was complete. They knew it would divide much free software from some free software and some open source.

There are very strategic things being done to fight free software. As with narcissism, the happy-faced campaign does not match the actions.

The playbook is public-- because it was made public, you can read Microsoft's plans to destroy free software. You can find them through Wikipedia. And it's time that people worked on some "Anti-Halloween" documents (in public-- because they too will be leaked anyway, and more people can work on them if they're public) to address the modern threats to free software.

Yes, the FSF will certainly respond to parts of this head-on. It's what they do. But although you can have free software without posix, if they destroy posix then you have lost a lot of the glue that helps hold the GNU/Linux ecosystem together.

When Torvalds retires, it is possible that GKH will take over. The Linux ecosystem is antagonistic to free software to the point of rewriting history (or gently coaxing or at a minimum allowing others to do it for them.) It is too cozy with Microsoft and it smears (via Torvalds himself) all free software advocacy.

That's not a minor problem. It should no longer be called "GNU/Linux," but as long as Stallman disagrees, of course we will probably all call it GNU/Linux. Personally? I think we gave them a fair chance at a compromise and there's no shame in rescinding the offer. The actual name of the operating system was always "GNU." The "/Linux" was a courtesy, a concession, a compromise. But as RMS is the one who made it, he may certainly disagree.

We can't control what other people call it. But that doesn't mean it's a bad idea to ask people to call it "GNU," because when they do (to this day) it shows what matters to them. It works.

But don't think just because Torvalds is giving GKH the hardest time of anyone, that he isn't grooming a potential replacement. And understand that politically, when GKH decides what goes in the kernel, it's probably going to get worse, not better.

Of course the FSF will compile its own kernels, everyone knows that. Non-free software won't make it into Trisquel. But if the design of the init can become more hostile and less supportive of freedom, so can the maintenance of the kernel. GKH is not an ally. Torvalds isn't either, but even if you don't like him we might all miss him when we meet his replacement.

Speaking of replacements, how is the clone vat doing? Like him or loathe him, Apple had only one Steve and we have only one RMS. Do I think RMS is infallible? No. Successful? Beyond his own dreams and some of ours. Clever?

When I was a kid, I thought Edison was purely amazing. Edison was in so many ways, the Gates or Jobs of his day; he took ideas and turned them into money and advanced the state of technology. To say the least.

But apart from the fact that this was not philanthropy-- it was not always advancement. Many years before Bill built an empire on them, Edison practically invented the EULA. And He controlled the film industry with patents, of all things. And with actual thugs. I am not a fan of Edison or Bill Gates anymore-- but let me say that Richard Stallman (for any faults he may have) is more like the man I wanted Edison to be than any other living person I can think of.

His story is much more interesting if you think of him as the person who (more than anyone) has spent most of MIcrosoft's existence standing up to them and building forces against them. So it is wise not to underestimate him. I try not to. But he is still human, and there is still only one. So if we cannot clone him (and my hopes are not high) then we need more people like him.

My list is Ben Mako Hill, Kat Walsh, Alex Oliva and Denis Roio. These are the four people I know of that I think are most likely to step up (and Oliva and Roio are unlikely, from geography alone-- also other obligations) if Stallman isn't still doing this at age 80 or 85. Perhaps you know others.

More may come along, and more ought to-- let there be more than this number already waiting to lead free software against its greatest challenges yet.

Because creating GNU was just the beginning. There are many, many more things that free software can accomplish. We finally have a viable, AGPL alternative to YOUTUBE.

You don't ever need Google to host your videos ever again (Obviously, you won't find the VEVO or Sony labels coming to PeerTube. So that use of Youtube hasn't changed.)

Mastodon had potential. Unfortunately, the community wanted to control speech more than it wanted to give users tools to do that themselves-- so depending on the instance, it is more like Reddit than the killfile in your email client.

I think there will be an increasing number of problems-- even more than that, an increasing number of opportunities, that the FSF will feel unable or unwilling to devote time or resources to.

To capture these opportunities and deal with these problems, I recommend "building out" the free software movement into more organisations.

The FSF will not be restructured. The other organisations do not need to be officially recognised, (the community can decide) nor do they need to compete for monetary donations.

Full disclosure: I have my own minor free software organisation. It does not compete with the FSF for monetary donations (it does not accept monetary donations at all.)

What can your organisation do without monetary donations? All kinds of things:

* Research and practice new ways to advance free software

* Offer software and/or documentation

* Provide educational tools, tutorials, and hosting / social interaction

In the United States for example, unless you file a return you need to be certain that your organisation does not have assets that reach or exceed $5,000 or that your organisation does not dispose of anything reaching or exceeding $2,500. Those are the numbers I recall, do look into this if your organisation has any assets at all.

Starting a non-profit corporation (501c or not) is cheap. Keeping it running is at least more expensive (in time or money.) But if you don't collect dues or have assets in it, you can probably run a small computer club (a regular meetup in person or online) without incorporating.

Organisations can also work together. Like two cores lending CPU power to the same application, two small organisations with a related cause can focus on different aspects of the larger goal. One can focus on making free software more fun-- another can focus on fighting corporate FUD. Still another can address things that are lacking in modern computer education.

Without any real budget, the most you can do is provide something with a name, and your time-- and time from your members. Theoretically an organisation should be able to run on volunteer time only, though if you are doing anything that requires insurance... either way, maybe look into it.

Operating as a non-profit with a budget and a number of members that have to pay to join, the FSF is likely to focus on what it can afford in those terms. SFLC is a 501(c)(3) and helps free software developers, so if you want a budget and you want to file, you can still start a 501c organisation if you have strict requirements for membership and meet other criteria.

But personally, the only "donations" my organisation is looking for are the occasional time, feedback and (especially) free software and other libre works (such as free-licensed OER materials, writing, music, graphics, websites, etc.) Even links to such works are appreciated.

More fansites for free software would be great. A well-maintained (not full of dead links) listing or webring of such sites would be great. I realise webrings are no longer in fashion. But our command line from the 1970s is useful.

Chapter 4: Teaching Everyone How To Code

In the decade that the FSF was founded, computer education was not yet based on applications. By the 1990s, education was moving towards application training, which meant two things: computer training became a lot more superficial, and it better served the market for proprietary software.

Computers are multi-purpose machines, and applications focus on specific tasks. This means that if your education shifts from teaching about computing to training to use applications, you also move from teaching something multi-purpose to teaching something application-specific.

This is fine of course, if all you intend to do with the computer is use those specific applications. This point should bother every free software advocate. **We are trying to give people control of their multi-purpose machines back, and they aren't even taught what they can do with that control.**

The essence of computing is not applications, but code. Although it is reasonable to assume that most people will not become skilled application developers, the fundamental understanding of computing is still missing for anyone that hasn't learned how to code.

Coding in one language to some degree teaches much of what someone would have to learn to code in other languages. So when Silicon Valley initiates their teach-everyone-to-code schemes they are gambling with the compromise that was made to education in the 1990s.

If everyone learns to code, then everyone gains some understanding of how to code in other languages. To a small degree, they get back a part of their understanding of what power they really have.

Nonetheless, education is still focused on teaching a lot of proprietary software. If free software advocates make it a goal, there is no reason we can't create "free software coding schools" (they will be cheaper if they're virtual. Consider something less like DeVry and more like Khan Academy) and stand up to the non-free-laden schooling that teaches people to compromise their freedom long before they're halfway through university.

We have such classes online-- we don't have our own schools, and one should be built. If someone can build PeerTube, we can make Free Software Academy and send all of our friends there.

If we do not reach at least high-school-level students with an education in free software, then we have squandered an opportunity to teach about freedom at an optimal stage.

If the idea is to reach people as early as possible, then a practical language that is easy-to-learn as possible should be considered. Since I have spent many years exploring this idea, I will share some of my thoughts about implementation.

First, I don't think a single implementation is the answer. It's a nice goal, but if I had a team of 20 people to work on such a thing I would split them up into 3 or 4 teams to come up with 3 or 4 different solutions. Then I would go to each member privately and ask them which solution they thought was best, and second-best (this means they must vote on at least one solution that is not their own) and I would ask them to explain their choices.

Perhaps the team could then work on the top two choices. I would also like for developers to try teaming up with educators (or vice versa) to develop teaching environments that are closer to what educators really need. This is a great opportunity for volunteers. Teaching this sort of computing to educators would also be a great idea.

Of course I don't expect the FSF to do everything. It only has so much money and so many volunteers. So this is a specific area where I think additional free software organisations would be useful-- whether the unincorporated, no-dues no-budget volunteer-only sort, or the more traditional 501c-type organisations (or both.)

But along with Free Software, Free Culture, Free Hardware and OER (I would prefer "FER" but this is ground that free software has lost to the word "Open" because they have not done quite enough to promote free culture, despite the obvious connections and the "free" in "free culture") society and free software alike would benefit deeply from an organisation dedicated to free software (coding) and free culture in education.

You may have noticed that a number of educational languages exist. This is one of my favourite topics, so I will talk about some of the options and why I like or dislike them.

First, any progress that is made with drag-and-drop coding or other existing solutions is absolutely great. Drag-and-drop coding is a very easy way to code, although we didn't need it in the 80s. Apart from that, text-based code is easier to share and it is hard to get a lot of people to take the idea of coding seriously when the primary objective is to move a cartoon cat around the screen.

I get it-- you get it, and sometimes kids get it. I am a fan of the MIT Media Lab and of Logo and of constructionist education-- I would like more languages that have the simplicity of Turtle graphics and the flexibility of Basic. And for years (decades really) I wondered what that would be like, because I was sure it was possible.

One of the best examples of drag-and-drop programming is App Inventor. I am a fan, this can be taught in high school, and it is certainly practical. I have owned several Android devices (several cheap ones and one very "nice" expensive phablet sort of device with a high-quality touchscreen) and I ran F-Droid apps on every single one of them.

I even got a touchscreen laptop to try out the sort of Javascript-based "apps" I made for the that laptop and for my phablet, but I finally realised that I hate touchscreens-- I hate tablets even more, and I really hate Android altogether. I have probably spent 10x as much time using netbooks vs. Android.

Thus, App Inventor may not be the best drag-and-drop programming tool we could use. Something like it, which also did "standard" GNU/Linux applications could be better. I would recommend something simple-- something that did not require state-of-the-art hardware, which ran in the browser, used javascript, and was stripped-down/light compared to something like Bootstrap.

I always recommend lightweight applications for education, because even if **your** school has plenty of money, countless others don't. As long as we are creating our own software, we should be standing against Wirth's law. I realise that Javascript in the browser is not always the most efficient choice-- which is one more reason that I don't favour it, but I did want to put it out as an option.

If your Javascript solution has a backend, now it can do some nice things I would prefer every coder be able to watch demonstrated and try for themselves-- like loading and saving files and loading html/files from other sites that allow it (not for layout, but for input.) I am familiar with Python, node.js and CoffeeScript, and personally favour Python whether used as a backend or by itself.

The first application I ever used on a computer was PC Paintbrush, at a time when almost nobody owned a mouse. This is what got me into computers. The second "application" that kept me interested in computers was a line-numbered version of Basic with graphics that went as high up as EGA. When I switched to QB, I gained basic VGA capabilities and a relatively friendly, GUI-like text interface.

Since I mentioned Constructionism and the Media Lab, I was very curious what the Sugar platform was like, and because of Sugar I tried Trisquel when it was fairly new. In the land of Basic, we often mused about creating "an operating system" (just a user shell, but we were kids) and Sugar is definitely Python's answer to the "Basic OS."

As with early days of web browsing and the modern web browser too, many elements of Sugar can be accessed and modified with a "View Source" option. Though I feel the environment is slightly slow and heavy (at least it was at the time) I think that has more to do with the design than the fact that Python is interpreted-- which I'm sure is a factor as well.

But the "View Source" feature of Sugar is brilliant, and what's more, my favourite Sugar "Activity" (I suppose it's better than "App") by far, is Pippy. I owe Pippy a great deal.

Pippy is the true QB IDE of the free software world. When I was a fan of Basic, for 25 years at that, I was also migrating to GNU/Linux (I had finished that long migration about the time I was exploring Sugar) and spent years trying countless dialects of Basic looking for the ultimate "21st century" solution to Basic coding. During this search, I learned to code in Javascript (for my purposes at least) as well.

When I found Pippy and Python, only Pippy demonstrated how friendly and powerful Python could really be. I immediately set out to separate the capabilities of Pippy from Sugar (not the application itself, but the Python programs) and I learned how to import Pygame and all that.

Yes, syntactically Python is a little more demanding than any reasonable Basic dialect. But the "feeling" I only got from coding in Basic that I knew I would find in a replacement, was there. Python truly gave me my childhood back. Learning it felt like the time I first started learning Basic.

Python is already widely used in education, as Basic was in the past. It is a very good language for education. But in order to promote free software, I went around telling people about GNU/Linux and demonstrating Basic, Javascript, Bash, as well as Python-- and every time I worked to teach fun Python applications to a friend-- it came up a little short.

I don't really think Javascript is a good first language for most people-- it is a great second language (even if you learned it first) and I think Python is a better choice.

But both are case sensitive, and Python has the (sometimes wonderful) left-hand whitespace thing... teaching Python (even with Pippy) isn't necessarily as friendly as we can get.

I took everything I learned from these efforts to teach coding, and put the experience into the design of a simpler language. And to be sure, I was trying to create both the simplest language I was able and (if possible) the simplest language ever. But I firmly believe the way to reach that goal is if more people try.

I never took a course on writing languages, I tried to create one in Basic around 2003. And I kept trying; I even made a fun language in Bash to make it easier to learn how to code in a language that Python or Bash would translate into Bash scripts. (It wasn't hard. I just kept it simple.)

I believe the iterations paid off, because I eventually wrote fig in Python 2. There is a Python 3 version, but I don't really like it. I knew a Former Nokia developer who also preferred Python 2. I am a fan of PyPy, and fig works great with PyPy if you change just two lines of code.

But if there were more efforts with similar goals as fig,

(I was not very familiar with tools related to Docker back then; though their Fig is mostly abandoned for Docker Compose now, as far as I know.

fig was originally named "fig basic" and someone whose opinion I respected recommended I drop the second part of the name. I actually chose the name based on a logo that the artist who made the QB64 logo made for another language of mine-- which didn't get around to using. That logo was a fig leaf...)

I would happily promote other free software languages with similar goals and similarly lightweight features. I don't think we have written the world's easiest language yet, though to try to teach people how to write programming language, I recommend writing a "hello world language" first.

A "hello world" program is a program that says "hello, world" on the screen, which is perfectly useless but introduces programming just the same. So a "hello world language" is a language that can only be used to write a "hello world" program... it shouldn't take very long to write one...

Shriram Krishnamurthi teaches people how to create programming languages at Brown University, and says that it's better to write a new programming language on purpose, since you could create one accidentally by starting with a simple configuration language-- and that's the worse way...

But after I repeat that I think there are lots of options for this purpose, I will outline what I consider a very simple language for teaching:

1, The language I offer as an example (fig) is implemented in Python 2. It can be used with Python 3 in a pinch, but I don't prefer that at all

2. Because Python is one of the easier "professional grade" languages to learn and use, a simple language implemented in Python 2 makes it very easy to transition optionally to Python, or to easier to modify it into your own language

3. fig allows inline Python, separated this way:

```
python
    print "this is python code " * 5
fig
```

4. fig uses Basic/Pascal(/Bash)-like commands to close multi-line commands, is not case-sensitive, and does not require indentation (except in optional inline Python sections)

5. there are fewer than 100 commands, ranging from rudimentary graphics to getting information from websites, to opening and closing local files and changing the text colour using ANSI escape codes.

6. fig runs on the command line in GNU/Linux, BSD, MacOS and Windows. I ran it on Android once, that was tedious.

7. Pygame is optional. If it is unable to run Pygame, fig does 16-colour graphics using ANSI. The Colorama library (or a term window that is ANSI-capable) is required in Windows.

fig is designed to be a simple, fun language that can be used for silly beginner applications, creating utilities, remastering GNU/Linux distributions, computer art and more.

But it is designed to teach:

1. variables

```
this_variable = 5 ;

that_variable = this_variable

p = "" : arr ; times 1000                    # 1000-item list, all strings
```

2. input

```
that_variable = lineinput ;
```

3. output

```
sometext = "hello world" ; ucase ; colortext 5 ; print
```

4. basic math

```
height = 54 divby 3 plus 7 ; print
```

5. loops

```
for p (1, 10, .5)
    now = p ; print
    next
```

6. conditionals

```
ifmore (p, 5)
    now = "p is more than 5" ; print
    next
```

7. functions

```
function helloworld
    now = "hello world" ; print
    next
p = helloworld
```

To compile a fig program:

```
$ fig hw.fig # you can rename or copy fig46.py to /usr/bin/fig
$ ./hw.fig.py # fig adds .py but you could rename this "hw"
```

Nearly all the punctuation/syntax in fig is optional; only "quotes for strings" and # hashes for comments are required. fig ignores the = and ; and also allows |

```
nl=10 ; chr
introducing_bash="ls | grep fig" | arrshell
now=join introducing_bash nl | print
```

So this probably isn't your cup of tea-- but like I said before, with input from other educators you might design the best educational language ever.

For the same reason I reject the idea that school (especially mandatory introductions) should just teach proprietary applications (as used in industry) I reject the idea that simple educational languages are a bad place to start.

They can make it easier to learn the fundamentals of coding and transition those interested (or required) to learn more complex languages, and earlier languages can be more forgiving of syntax errors if there are fewer places to get the syntax wrong.

Maybe you don't want to go the "optional syntax" route. The goal was to create a Turtle Graphics like language with as little syntax as possible.

My first compromise was to add an optional colon for Basic-style command separation. I added a semicolon as well. I added a few others, including parentheses. Even if these are not required, they help the coder learn and think about syntax even when it's not required. They also help make code look nicer and more organised, sometimes.

I have (recently) written a friendly introduction to computing concepts that includes a simple introduction to coding using fig. It mostly focuses on concepts, not code. fig and this book (and the book I just mentioned) are in the public domain via the CC0 waiver.

I have also read the warning of Bradley M. Kuhn that public domain software feeds proprietary software. I am not sure what percentage of the time that is true, but it is a lot less than 100%. I understand the goal of free software to produce copylefted works, and that isn't a bad idea. I have read the FSF pages on when to use the LGPL, etc. and I am pretty certain there is a lot of trivial code out there (vs. elaborate applications and operating system components) that doesn't feed Microsoft and Apple.

For elaborate works, I certainly understand why free software would want to go copyleft with GPL 2 or later, or the AGPL.

I got into fully-free distros via Trisquel, and into Trisquel and Python via Sugar. I got into Debian via their stripping of non-free code from the kernel, paired with a bug that likely destroyed one of my drives years ago.

There was a problem with power management and some types of ATA drive, which Debian was swift to counter with one or two lines of imperfect code (it just called hdparm on startup) and the distros that decided to do nothing left unsuspecting users to watch their operating system destroy their drive.

I am certain the number of victims was few. Nonetheless, I was watching the start/stop count spiral off the chart, and when I applied the fix it calmed down (though it was too late.) Back then, I did not have a whole lot of drives, but software is imperfect and what can do you?

Nonetheless, I noted that Debian took care of this and other distros opted to WONTFIX and that sort of thing. The reason was the fix wasn't perfect and "might not" work on all systems. Well phooey, I switched to Debian. It was closer to libre than what I used prior to Trisquel and it wasn't destroying my drive. It wasn't that I couldn't apply the same fix to a Trisquel machine-- it was that no one else was willing to do anything about this. Just let the users hardware die. So I left those distros behind. This was many, many years ago, and I was happy with Debian for quite some time though I continued to try other distros.

When I finally left Debian, it was after rc.local stopped working and I couldn't figure out why. I wasn't running stable, so at first I was like "Hmm, alright. Well, this is odd but it's not stable, I guess this sort of thing happens."

Note I was running stable on all machines except one-- because I chose to run a single machine with a more up-to-date Debian, I got early warning about the thing that was stopping rc.local from running.

When I found out it was a different init system that had installed without warning-- I mean, Debian updates tell you when there's an update to foreign time zone info, why (other than politics) would they not pause the installation to let you know that they are removing and replacing your init? That was just shoddy.

I learned everything I could about what was going on, and decided to leave Debian over it. It was a few months before I found a replacement for what was once the most reliable operating system of all time. I still can't believe that the FSF isn't treating it as a threat-- however, the damage is done and even if the problem went away next week, free software was set back for at least 3 years by this. How can the FSF not realise and condemn the goals and arrogance behind this thing? They too are affected.

Either way, Before that happened, I was promoting Debian by putting it on machines that people didn't want anymore (I would get them free) and then giving those machines away to people having computer problems. "My computer isn't working." "Here, try this one." "Hey, it works!" "Keep it." I wasn't the first person to do this, but this is the easiest way to get people to use free software.

https://freemedia.neocities.org/zero-dollar-laptop.html

Although I wasn't using an FSF-approved distro, I was getting them away from a fully-non-free OS, and I rarely installed Adobe plugins (I loathe them and don't touch the things on my own systems) nor did I ever recommend the non-free repos (which typically weren't added.) So if you aren't going with a fully-free distro, this would be the next-best thing.

Since I no longer believed in (or wanted anything to do with) Debian, I needed time to let the alternatives come up to speed. One thing that I really liked about Debian was that I could download their source DVD, copy it onto the system I was giving away, and satisfy the GPL without gobs of server bandwidth or burning large DVDs. Install, copy the source, DONE!

Debian was nice about that. I'm afraid most distros don't make it that easy (Yes, Trisquel is good about this.)

I really have to say, that if it is your desire to distribute operating systems with only slight modifications (and not to the source but to binaries) that most people still (even those that care) still do not understand how to do this. Most communities that produce distributions are very lax about it. Those that are strict are sometimes just as unhelpful.

I have added Bradley M. Kuhn's guide to copyleft:

https://copyleft.org/guide/

to the Free Media library, but even with distributions like Tiny Core Linux which were born of a desire for greater GPL compliance (Tiny Core rather than DSL) it is not always obvious to people how to comply other than "compile it yourself." I realise there are people who understand how this works, but let me explain what I mean:

Let's go back to my example about Debian. I used to distribute Debian on gratis hardware installed from their LXDE hybrid (CD/DVD/USB) ISO. The source DVD had the same name (close enough) with "source" or "src" in the filename.

If I installed Debian this way, I felt with confidence that I was complying by also copying the source DVD to the hard drive. Don't want the source? Delete one file. Alright, so now I'm a good (pretty good) free software person.

Props to Trisquel-- when I did this years later with Debian, I got the idea from back when I had gotten a Trisquel source DVD with my FSF membership card.

When stopped using Debian, I was hoping to do the same with Devuan. While I was migrating and waiting for Devuan to mature to that point, I worked on other things. This included fig, so that I could continue to teach computing and about free software while my primary operating system was being overhauled and repaired.

But for a system like Devuan it is not so easy to comply as with Devuan or Trisquel. I once asked RMS himself (and he's probably going to say I'm misquoting him. I thought I understood him and I'm doing my utmost to get this right, but if I misunderstood him it's certainly not deliberate) if I am violating the GPL by helping people install new packages after going to all this trouble-- in other words, if i need to download the sources for them too.

To the best of my knowledge, I can setup a system as I described doing with Debian, then give it away, then help them install packages. It would be nice to know a full list of my options. I have spent some time looking through the aforementioned GPL guide and I am certain it covers a lot of this-- but it is 150 pages when I copy it from 1 pg HTML to OpenOffice, and I did note that Part 2 starts with Chapter 13, so perhaps I should read from there first before going further.

But when it comes time to modify the Tiny Core distro, no one thinks this is "necessary." Personally, I've had as much fun remixing distros as editing program code. In fact, I don't remix software "by hand." I write scripts that automate downloading, opening, editing and creating bootable ISOs from existing ISOs. And as long as I don't distribute the resulting ISO, I am not violating any license.

How to make it so I can comply enough to be CERTAIN those ISOs can be distributed is where I get confused.

And no matter how much you know about this, or how obvious to you, this stuff is confusing. It matters to some of us. Those of us who care the most are generally asked to decide between "Don't worry about it" (from those who don't worry about it) or "Don't bother remixing that particular distro then. Why bother? There are plenty of fully free distros you can use."

I am not asking for the free software community to take my favourite distro and "make it libre" for me. What I am saying is that if I want to do that, I should be able to do that with free software. I should be able to take Ubuntu, remove all the non-free parts, throw in a libre kernel, and have a libre distro.

I am told "it's not that easy" but that isn't what bothers me. It's the licensing (of the GPL parts) that I want to understand, and it's complicated for me. Rather than expect the GPL to be less complicated, I would really like to understand better. But a lot of this isn't new and this is one of the major challenges facing free software. It is absolutely easier to remix an operating system than it is to comply with the licensing.

And I don't think the free software community "owes me" a better understanding. They may not even like what I'm doing with that understanding. But there is a lack here, which someone could do something about if they didn't want the lack. And there are people who are more confused about this stuff than I am-- People with good intentions who think if they just provide a mirror of the unmodified Tiny Core distribution, that they are now free to distribute their modified version.

So much of the community of free software users (who provide support and even friendship to people using free software) don't know what they're talking about when it comes to this stuff. They enjoy remixing distributions, but the licensing goes over their heads.

If I were part of the BSD community I might just say "Let's ditch the GPL. Let's migrate as far away from it as we can." And to be entirely honest, I don't think it's worth that.

Instead, between free-software libre compliance and what Kuhn calls "the most egregious violations" of GPL, including selling hardware with modified GPL-licensed software and no source code for even the modifications (let alone the originals), most re-uses of free software are in this huge, growing "grey area" of non-compliance.

I am not content (I just can't make enough out of it) to be on the "Let's just use Trisquel" side of this. Nor do I want to be mired in non-compliance with the rest of my hobby-oriented compatriots. I have done "the right thing" when it was trivial, and I have tried to make progress in my understanding and my deeds when it is less trivial.

Still, I find this is very uphill. As good as it is to set a perfect example, I feel that free software has this huge opportunity to create a small organisation that is devoted to working with "the middle" and growing, hobby-oriented sector of this non-compliance. I think it is (sincerely, and not this is not a critique at all) beneath an organisation like the Software Freedom Conservancy. I admire their work, but this calls for something more casual-- something that can devote more time to "small steps" towards progress, and setting better and better examples rather than setting perfect ones.

And I repeat that free software doesn't "owe" the people who exist in this grey area. Similar groups exist within copyright compliance-- you have people who try to never infringe, people who do not care at all and are happy to infringe, and I would say most of society (if we are honest) is in between.

There is a great opportunity to coax a lot of well-intended people upwards in this, which I honestly doubt appeals to most free software advocates. It appeals to me, and if I ever understood these matters as well as Kuhn or RMS, I would not be leaning towards the "Why Bother? There are plenty of fully free distros" stance.

Nor would I lean towards "Don't worry about it" or "Let's just migrate from the GPL." I really like the software that's under the GPL license. I am not, as the BSD community seems to be, interested in replacing as much Copylefted software with non-copyleft as possible (though I do tend to put my own software in the public domain. I want strict beginners to be able to share and modify it with zero apprehension.)

I feel it is probably a waste of time to appeal to the free software community about this, but I do hope to eventually find people who share an interest-- wherever they are or whatever they're doing. And whether they like their own methodology and think it's better to just set a perfect example or not (it often is) I think it will give free software a giant boost if the people "in the middle" are slowly taught to care more and understand better. There will always be people who don't care at all. But every day I am around people that care more than they understand.

I doubt I could put it much more clearly than that.

Chapter 5: Other Paths with Education and Free Software Advocacy

We teach science to everyone not just so they can become scientists for a living, but so they can understand the world they live in. We should be teaching (easier, high-school level) computer science to everyone for precisely the same reason.

Code (done in the simplest ways possible) is the most obvious interface for teaching those concepts. Code is merely a symbolic abstraction of what computers do-- computers are merely a functional implementation of what code does.

Code and Computers mirror each other to the point where you can implement a CPU as an application, and then run the same operating system on that application (such as with Qemu) that was designed not fo the application, but for real hardware. We can talk about Turing, but this mirror of Code and Computer is the essence of what he proved to the world.

If the Free Software movement wants a new generation to supports its efforts, we need to have enough people that understand how to code and it is certainly preferable if they spend more (or all) their time coding with only free software. I learned how to code with a proprietary language-- but it was a language that was so simple, I was able (without taking a single course in language design) to spend years guessing my way into implementing an environment that is similar enough for my preferences as the language I first fell in love with-- using only free software.

I was very proud to notice that people with a better idea of what they were doing used similar techniques and tricks as the string processing and lookups and even the conditional structures I used to implement my early "language" tools with-- I mean, computers are very simple until you get into the very specifics which are **most optional**. Those things that are harder to understand are generally the ones that vary from Intel to AMD, CISC to RISC and 16-bit to 32-bit.

In other words: **options**. The fundamentals of computing can be so much easier than the optional aspects of the modern implementations. Not that implementing a dialect in Python isn't modern, just that it needn't be sophisticated. You can start implementing your first language before you fully understand programming, or know much about writing excellent programs-- and I've explored this as a way to teach coding in the first place.

1. iterate through lines of a file

2. iterate through bytes of each line

3. iterate through tokens derived from bytes

4. process those tokens using conditionals

That may not be a recipe for a great language, but it's enough to teach a beginner how to experiment and think like a language author, not just a coder using someone else's language.

I firmly believe that if we devoted more of our effort to computer education with free software-- not just teaching people "why you should use free software" or "why you should use free software in education" (of course you should!) then we are going to have more free software authors and many more free software users.

Since I have already put it into the public domain, I will include a few concepts from my book on "holographic learning" in this chapter. This is probably all stuff you know, but this is stuff you could include on a file in your favourite distro called "Learning more about computers.html"

As outlined in Chapter 2, the BIOS (or EFI or UEFI) hands the computer over to the bootloader, which hands the boot process over to the operating system.

The operating system runs an "init" and/or startup scripts to get the OS running and to present the user with an interactive environment (or, to run a server that can be accessed from another computer.)

It is at this point, between the operating system and the interactive environment, that we can begin to talk about user applications. And the purpose of doing so goes far beyond simply how to *use* the applications.

The interactive layer of the computer system is called the shell, and the shell runs applications.

For a command-line shell, the way to run an application is to type its name. For a graphical shell, you might open a window or menu that has a representation of the program-- an icon or a name for you to open by clicking on it.

Whether the program name is clicked on, typed in or spoken out loud, the shell recognises that the name is referring to something installed on the system, and it tells the computer to load that program.

Running a program and calling a function inside a program is a very similar thing-- a program may contain several functions, such as the **to_f** function we showed the code for earlier, and the function will "run" when the program refers to it by name.

Putting a file that has runnable code on the system will often let you run it by referring to its name. So using the command line is a lot like (or is) an act of coding.

When is a command less like coding, or a function call? When it's a query. If you tell the computer the name of your browser, whether you type the name or click on the name or speak the name out loud, this is a lot like a function call. It runs the code referred to by that name on the system.

The data you type into the google search bar is called a "query," which is a bit less like coding, because you aren't referring to the function you want to run-- you are simply entering data which Google hands to the function or functions for you.

But ultimately, every process a computer (or internet website server) can perform can be thought of (or involves) a function call.

What is a function call? It is simply running a specific section of code that has a unique name.

The graphical shell, like the computer itself, tends to deal with rectangles-- numerically, these rectangles may actually be straight lines:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

And the computer might not care if each discrete point in the line is arranged in a row, like this:

```
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19
```

Or if the points are arranged in a rectangle, like this:

```
00 01 02 03 04
05 06 07 08 09
10 11 12 13 14
15 16 17 18 19
```

We think: 5 points across, 4 rows down-- the computer may think "20 discrete points." There is pretty easy math to convert back and forth either way. The way these points are numbered and referenced is called "addressing," like it is with post boxes.

But whether the addressing is in a line or rectangular, it is basically a fact that the graphical shell deals with rectangles. Your screen has a discrete number of points across, and a discrete number of rows down, and each point is called a "pixel," for "picture element."

When you click on certain areas of these pixels, the computer is told to call certain functions. This is the basic concept of how a graphical shell works.

GUIs have different design elements, such as arrows or a bar you can click on to scroll, menus you can click on that draw a box and put different items in a list for you to select, buttons you can click with the mouse to perform a certain task, or text boxes you can click on and then type text into.

In a web browser, the browser contents can be defined with document "markup language" such as HTML. HTML can be styled with CSS, and various aspects of the page can be defined, altered or controlled using Javascript. Using these languages, you can define the elements of a GUI within a webpage.

While a Cartesian graph consists of two such number lines, including the other half for negative numbers and a vertical number line (also centred at 0) creating four quadrants of the graph, in computers the points on the screen are most often represented only with positive integers or whole numbers. (and most often including 0.)

```
0 --|----|----|----|----|----|----|----|----|----|----|----|---- >
 |                              (x values)
--
 |
-- (y values)
 |
--
 |
--        A typical computer graphics scheme
 |
V
```

Instead of using number lines, we can also use coordinates in a rectangle to represent the points:

```
(0, 0)  (0, 1)  (0, 2)  (0, 3)  (0, 4)
(1, 0)  (1, 1)  (1, 2)  (1, 3)  (1, 4)
(2, 0)  (2, 1)  (2, 2)  (2, 3)  (2, 4)
(3, 0)  (3, 1)  (3, 2)  (3, 3)  (3, 4)
(4, 0)  (4, 1)  (4, 2)  (4, 3)  (4, 4)
```

A difference between graphics and text is that in terms of addressing, it is common for the top-left corner to be addressed as (1, 1) rather than (0,0):

```
(1, 1)  (1, 2)  (1, 3)  (1, 4)  (1, 5)
(2, 1)  (2, 2)  (2, 3)  (2, 4)  (2, 5)
(3, 1)  (3, 2)  (3, 3)  (3, 4)  (3, 5)
(4, 1)  (4, 2)  (4, 3)  (4, 4)  (4, 5)
(5, 1)  (5, 2)  (5, 3)  (5, 4)  (5, 5)
```

So, if we are covering the top line of our 5x5 screen graphically, it will be with a line of pixels ranging from (0, 0) to (0, 4):

```
(0, 0)  (0, 1)  (0, 2)  (0, 3)  (0, 4)
(1, 0)  (1, 1)  (1, 2)  (1, 3)  (1, 4)
(2, 0)  (2, 1)  (2, 2)  (2, 3)  (2, 4)
(3, 0)  (3, 1)  (3, 2)  (3, 3)  (3, 4)
(4, 0)  (4, 1)  (4, 2)  (4, 3)  (4, 4)
```

But if we put the word "hello" at the top, it will probably be from (1, 1) to (1, 5):

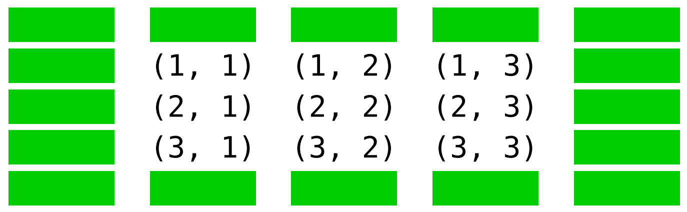
```
h       e       l       l       o
(2, 1)  (2, 2)  (2, 3)  (2, 4)  (2, 5)
(3, 1)  (3, 2)  (3, 3)  (3, 4)  (3, 5)
(4, 1)  (4, 2)  (4, 3)  (4, 4)  (4, 5)
(5, 1)  (5, 2)  (5, 3)  (5, 4)  (5, 5)
```

Extra points if this stops you because you noticed that in the top graphical example, we have created a large block for each "dot" and in the second, text-based example, we have used characters such as "h" that have a shape which clearly requires more than a single pixel.

We can certainly do a graphical or "pseudo-graphical" display of graphical data using large character-sized blocks of the screen, but "graphics modes" on a computer generally have finer resolution (for example, HDTV does up to 1080 rows from top to bottom, and a larger number of units across) and text requires more than one row and column of pixels.

At the moment this line is being typed, the screen is at a resolution close to 1920 x 1080, and the characters (if we take a screencap, zoom in, and count the pixel dimensions of this character: █) are actually closer to 16x24. We should be able to get about 5,400 of these characters (1920 / 16 * 1080 / 24) on the screen at once, if our font is fixed-width.

Getting back to graphics, our example screen can hold up to 5 x 5 or a total of 25 dots, and each one can be any of up to 16 million colours. For this 5 x 5 box, we will just use green:



Ultimately, the means of drawing a box like this is to address each dot and change the colour information for each of these dots. Just like the location of each dot itself, the colour is represented and stored numerically.

Without getting into the way colours are converted into numbers and displayed as colours, [that would be a great addition, I just chose to skip it] we can say that everything that happens visually in a graphical interface consists of changing the values of the numbers that store the colour of each pixel.

Before, we said that everything that happens is a function call, and this is also true-- every function call references or writes the values of some of the numbers stored in the computer. To get information from a file to the screen, one function may read numbers from a file and another may write numbers to the circuits that change the display.

Chapter 6: distro-libre, Reinventing the bootdisk and casually remixing free software

Let me make perfectly clear, something I was talking about in chapter 4-- if you can modify and recompile a GPL program it should be no problem for you (though somehow, it sometimes is for some people?) to include your source code and instructions/scripts for compiling it.

If I compiled all my own software, this chapter would be a friendly guide about how to compile software instead.

It's not that I've never compiled anything. Sometimes it's easy. I have a basic understanding of dependencies, I have modified and recompiled (but not redistributed) xtrlock to have no cursor. (It's really really easy. And I really don't even know c to the point where it would help with this, you just have to understand arrays/syntax in general to make this modification.)

I would have zero trouble fully complying with the GPL if I chose to redistribute that modified program.

One of the main purposes of this chapter is to get you to understand what it's like to be in that grey area of GPL compliance. Puppy Linux, for one of the most prominent examples, was once harassed about GPL compliance (whether by a zealous fan of free software, or just someone who didn't like Puppy and wanted to cause grief-- it could even be but I doubt it was both) And you can build Puppy using the Woof system.

I found the Woof system too complex to use and set out to remaster Puppy automatically, using a script that runs, downloads files, modifies them and puts them together again.

The entire process is automated, and requires no outside input unless something goes wrong. Usually this is because something like xorriso or genisofs was not installed, or squashfs-tools or isolinux was missing. The system uses Python and standard libraries only, so it can run from Debian, Devuan, Refracta, Void Linux, Puppy, Librepup and Trisquel.

You change the distro by changing the program-- so when you have it the way you want, you have an option:

1. Distribute the ISO, which most people will want to do of habit

2. Distribute just the program-- it is public domain via the CC0 waiver, and provides full, automated instructions for people to create you custom "distro" (derivative, spin, pup, etc.)

I suppose if this were setup as an online service (not my favourite way to do it, but as a way towards compliance) it could create the ISO (using about 15-30 minutes of server time for squashfs) and offer urls to the source for each distro used. Maybe this is naive-- cheap means of compliance are essential to all but the extremely serious hobbyists, and that server would be a second project altogether-- the rest can be done with a script!

But the option to just distribute a "custom ISO via 150k script" with NO VIOLATION of the GPL is really nice, and yet people still want their ISO download. If this script being available counted as "providing source" it would be interesting, but probably not the direction anybody really wants the GPL to go in.

Almost anything people want from a remaster is possible this way. You want a "new OS?" You've got it. You can put it together in a week. This casual attitude is alien to the serious portion of free software community, but I liken it to the days of the DOS boot disk.

When hard drives were not a given (and with typically just a 1 year warranty, they aren't exactly a given now) we actually did alright without them. DOS was a very small system designed to be "just enough" for running your programs.

Of course, "just enough" turned out to be a lot of "fun" when it came time to extend RAM, load graphics and sound drivers, and leave enough of that precious initial 640k to the applications that demanded or required it. Then again, you could put the OS and some of your favourite programs on a floppy and-- worried a friend might put it near a heater overnight? DISKCOPY A: B: and you can create a spare.

Doing that with a hard drive remains less trivial, despite the history of utilities (including dd) that make it easier to do so. But with a USB or DVD writer...

I actually "install" my own distro on one machine by simply running dd with the ISO as the "if" and the hard drive as the "of." Then I add a partition and put more files on it. I can't easily update the OS on that one, but it won't boot from USB anymore and the optical drive doesn't work either.

I can update most parts of that machine in other ways. But for probably every other distro I've worked with, I've figured out how to configure grub to load directly from the ISO file-- no dd necessary. (Just be certain you backup your grub configuration.)

It is just fun to create bootable media. you can put all your favourite programs on it and run it everywhere. You can write the ISO to DVD or USB. To this day, I try to fit it on a CD. I still find CD-only machines, and it is more likely to load into RAM with room for your OS to run on a 10-year-old-machine. CD-sized is not an arbitrary "vintage" eccentricity, it is still a practical limit for some things.

I am far from alone with this, and the thing about these boot disks is, they proliferate. For better or worse, we just keep making more. If you do this often enough, it can even serve as a backup strategy. But even though I once setup my bootable Trisquel USB with a persistent folder /partition / something-- a nice option that many people want-- what can I tell you? It's "not the same."

Settings / configurations and spins are related, but they're not equivalent. People like to change Defaults. If they don't, someone does it for them. They recognise the power in doing this, even if it is less than the power of building a full unique distro. It's "enough reward" for many, and much less work. Sure, people who actually create full distros aren't impressed by such things. But many other people are, and it makes them happy.

I do not know of any fully free distro that really facilitates this desire that many free software users have. I am not certain this desire is one that is widely understood or appreciated by (or necessarily even fully in line with) free software.

I am however, convinced that this can be better catered to-- once again, whether or not someone is truly interested in improving this area of free software use/reuse-- and that doing so will produce a better community that (on average) cares more, not less about the software freedom goals of the GPL.

I am not expecting this to go somewhere. But the other thing I believe it could do is help to bring more distros into GPL compliance. You may not think we need more distros, but there are many out there. Just switching to fully free would solve a lot of free software problems.

Ultimately though, it would not cover all of the things people use free software for. Rather than argue that not all things people do with free software are good for free software-- let's talk about the fact that people have wanted to create boot disks whenever trivial to do so, from at least 16-bit computing in the 1980s, to the present.

For whatever reason, most or all of the distros that make this worthwhile are not as dedicated to freedom as the FSF and Trisquel. It's probably because the people most interested in doing this are not hardcore FSF/freedom fans-- but casual users. But Slitaz, Tiny Core and Puppy can all be brought into (or exist in) full GPL compliance. I do not like their kernels (except for Librepup) but they are fun to use.

It is not the non-free software that makes them fun. For some, it is-- for me, it is the feeling of exploration, of progressing towards a system that is perfectly tailored (or close enough) to the needs of the individual user-- but not only the individual user.

That is a completely unsustainable goal with non-free software, and it is less sustainable with people who don't care about your freedom (I don't mean people who have grey compliance, I mean people who are openly antagonistic to your freedom, who redesign software you already like to give you less choice than you started with.)

This is something only boot disk culture can do, and I would like there to be a confluence of boot disk culture and fully free software. Librepup is the closest thing I know to that. We need more things like it.

I would like there to be a distro-libre project, where people work to make existing distros (especially boot disk distros) more libre by automatically removing components that are not free. For full GPL compliance, they can be distributed via script rather than ISO. But archivists will continue to put these ISOs up on the Internet Archive. I would love to have a lengthy discussion with Bradley Kuhn (or any other interested party, including RMS) on a way we can make this grey compliance a bit lighter and make progress.

Chapter 7: Moving past lost opportunities with Free Culture

The founder of Creative Commons is perhaps the world's biggest fan of free culture, but he was or is on the FSF board or steering committee. From going on the Colbert Report to suggesting we call an Article V convention, to running for President in 2016, Lawrence Lessig (winner of FSF's 2002 Award For the Advancement of Free Software) is a man I deeply admire. I keep trying to guess what he will do next to try to further freedom not only for software, but for all culture-- I suggested he try terraforming a planet with better copyright laws.

The story goes that Gandhi was once asked "What do you think of Western Civilisation?" And he replied, "I think it would be a good idea." I feel the same way about the Free Culture movement. The only reason I stopped funding the FSF was I grew extremely tired with this being plastered all over not just the FSF website, but some related sites:

This work is licensed under a C------- C------ A---------n-N-D--------- W---- 3.0 l------ (or later version) — ***Why this license?***

The "**Why this license?**" part is what irks me the most. The FSF promotes what I consider an alternative to free culture that I find as harmful to the movement as "open source" is to free software. I love RMS, he's one of the most important people in the world (go back to where I compared him to Edison and Bill Gates, except the hypothetical GOOD versions of those) but I absolutely can't tolerate his "works of opinion" shtick.

You don't need me to go into detail about that, I couldn't do better than Nina Paley's "rantifesto" about Stallman's take on free culture:

https://freedomdefined.org/Licenses/ND

My feelings about this have not changed in many years. Stallman is very simply **unfair** to the free culture movement about this. I don't think he deliberately misrepresents them (I don't know if he has ever deliberately misrepresented a single person in his life. He is honest to a fault, and that's admirable) though these apologetics for the most useless (and meaningless, and impotent) license that Creative Commons ever made-- a license they should have retired and started to discourage use of long ago-- are the only purely intolerable thing I think RMS does.

Of course he's entitled to his opinion! And people are even entitled to support him. It creates a huge rift between free software and people like myself, who agree with the gist of Paley's Rantifesto (and I felt that way before I ever read it.)

Whether or the FSF continues to endorse this license, the rift is there, and it is unfortunate. I feel strongly enough about it to deny the FSF funding, because I don't know how else to protest this in a meaningful way. Through me, free software has gained several hundreds of dollars towards various free-software projects. I don't feel at all like a freeloader. I devote most of my free time towards working out ways to make free software work better for everyone in the world-- and how to help the FSF indirectly.

But freedom is an exchange, it is a conversation. I acknowledge that the FSF can help in ways that are fundamental, but I do not simply give them all my agency and let them dictate how to be free. I say this cautiously-- I am not accusing the FSF of being dictatorial and I realise that opponents of free software like to pretend they are easier going, because they replace the FSFs written rules with unwritten ones.

But the truth remains that freedom is an exchange and a conversation, and until I feel like free culture has enough of a place alongside (not necessarily as part of) free software, I will always think "we are doing this wrong." I have devoted years to refining and reiterating this argument, and the last straw was when the FSF kept sending their membership letter-- also with the ND license.

Here is mail! You can't edit it! I'm aware of first-sale doctrine and my fair use defense, but honestly this was too much and it just flies in the face of reason for me. I know other free software advocates who feel this way-- I meet them here and there. We just don't buy any of the reasons that "Works of opinion" should carry a meaningful difference.

But this is supposed to be a constructive book, so I will mention the places I think there are progress and hope, and perhaps room for further progress:

First, I am very pleased that an increasing number of free software platforms are using real free culture licenses. My organisation really has no place for NC or ND, and I think where OER commits to free licenses they are making progress. I also have no time for the GNU FDL, not only because of invariant sections but because I don't think it is useful to distinguish between paper copies and electronic ones for a documentation license. In fact, I think it is harmful.

-- Sorry. What I meant was, The GPL compliance guide is CC BY-SA 4.0, which is simply great. And LibrePlanet put up lots of video under a real free license, which means that if you bother to create transcripts they will also be under a truly free license. And in one of those videos from LibrePlanet, Ben Mako Hill said something like: "We [may need to] distance ourselves further from Open Source."

Given that Open Source is the very womb of redix, I entirely agree. They have succeeded in selling so much free software out to the non-free competition, and proven their true motives. The illusion that Open Source cares at all about or has anything to do with Free Software should be finally debunked and proven false at this point.

But they still have their (better) support of Free Culture as a real feather in their cap-- and that means if you support free culture as well as free software, you can't help feeling torn between FSF and open source just a little, even if you find the latter completely loathesome.

Lessig should be taken more seriously by the free software movement, first of all. And so should the movement he took to a completely new level. You can say that free software (as formerly uncopyrightable during its early days) existed prior to the FSF. You can say that free culture (in many forms) existed prior to Lessig's involvement. Though you can't deny that Lessig helped found and define the free culture movement we have today.

The free culture movement is too weak and needs greater support, and if you believe in it at all, you should try to support it more. Not with money-- with time or **thoughtfulness** and above all, with free-licensed works.

My attendance at a free culture event in DC (where I met Kat Walsh the first of just two times. People spoke of Karl Fogel by first name only, I don't know if he was there or not-- Kat Walsh is the only individual from the FSF I've met in person, which is why I mention her a few times. I'm sure she doesn't remember me, which is alright-- it's not like we started chatting every day on Twitter. I've also met the developer of Refracta in person a few times) convinced me that there are people who "care" about free culture in the same way that open source "cares" about free software. And my feelings about free culture are very mixed, except for one absolute:

I want MORE free culture! And lots more, at that. We absolutely do not have enough now, and we need to keep growing that amount. So if you care about free culture, please-- create more free cultural works. As much as you put into a fixed medium and are willing to share, put some sort of free culture license on it. The Free Media Alliance currently recommends these licenses as best choices:

https://freemedia.neocities.org/recommended-licenses.html

Note these include both software and cultural licenses. "Licenses for documentation" do not need to differ from cultural licenses as they do on the FSF page. OER licenses do not (as a best practice) differ from documentation or cultural licenses-- OER (done right) is a subset of free culture. If you need an obvious example, Thank you again Mr. Kuhn for using CC BY-SA 4.0. (OER has many others.)

Chapter 8: What to do with this book

I would certainly be thrilled if you took some time to look through this book, and particularly if you find just one detail or argument that inspires you in a meaningful way.

I would be just as happy if this book turned into a longer (or broader) conversation between us, or between more people.

If you would like to refute parts of this book, I encourage you to do so. One of the main inspirations for this book at this time is the Trisquel community. I have always taken its warnings at face-value, and found them off-putting and dismissive. In short, I have made too many assumptions about the Trisquel community. I don't like to spend my entire life living on just assumptions, which means that when I find the time and patience and faith-- even if I have prior experience (and I do!) to support my incorrect assumptions, I continue to go back and check on some things.

I have over the years, been through several iterations of free software philosophy. I started with Open Source, and spent years trying to resolve its rhetoric with its deeds.

I came to the conclusion that Open Source is marketing above all else, and not straightforward or even as meaningful as it purports to be. It also confuses a lot of good people who buy into it, which is the obvious reason why not all Open

Source fans are bad people. I also think a lot of the worst offenders, including Torvalds, put so much into their rhetoric that they sometimes even fool themselves.

Linus Torvalds doesn't wake up in the morning and ask himself "How can I fool people into thinking it's 'Linux'?" To him, it absolutely IS "Linux." It should be more obvious that he's mistaken, though Open Source is deeply invested-- in various ways and meanings of the word "invested"-- in being mistaken.

Every person out there, lives in a pretty cynical world. For most, it is not a world of their own making, but one that was "helpfully" made for them to exist in. Not all help is false, not all disagreement is based on fallacy or misunderstanding, not all imperfection is worth fighting. But with the exception of narcissists and the like, every person out there is a potential ally.

Open Source wants you to STOP what you're doing and devote all your time to wooing the long tail (of people who don't care about your cause.) They want you to abandon your way, and do things their way.

If you take anything I've said here to mean that, just remove that part. Or reword it so that it doesn't say that to you. Or just keep the parts of this book (and delete the rest) that appeals to **your** own goals in some way. This book is designed to help, not to push you away from your goals. But that will only happen if it is understood and used for that purpose.

I invite you to use this book in whatever way suits **you** personally, as well as your own values. But I would be happy to live in a world where all software is free software, where all culture is free, and where our motives for doing things have absolutely nothing to do with monopolies or trying to unfairly or unreasonably control each other's lives.

Please feel free to remix, add, debunk, improve, or comment on any part of this book. I hope you will do so honestly, and thoughtfully. But, I have tried my very best to extend that to you. I doubt I have done that flawlessly, and I accept that you are not perfect either. It's something we all have in common.

Best,

figosdev

Sept. 2018

P.S. I've let some of my favourite draft versions go unedited for years. There were more things I wanted to say. One of the reasons these are short books is that they reach a certain point and then I can't stop thinking-- "You know what? Let's just put it up like this and find out what happens."

BONUS PADDING MATERIAL!

AI -- A futurist's interpretation of the present (not written for experts...)

One way to think of AI is "A lot more computing-- both good and bad." For art? Great. For surveillance? Sometimes bad. Apply it to everything-- people will. And it will be a great multiplier of things; of all computing tasks, more or less.

Not all at once. And I'm not hyping it, I'm describing the effect it will have-- as a multiplier:

Another way to think of AI is "enhanced computing." Because in many ways, it is fundamentally "just computer processing." Anything a computer does is "just computing." But with AI, this becomes something more incredible. The scope of what can be touched with computing becomes richer-- for good and for bad.

Computing is very flexible, by design. We can actually say something about AI while being this vague-- it is essentially like computer processing, except that it can do a little more, it can do more with more modest requirements-- it may take a while-- but with home computing equipment you can suddenly do things that you would expect of companies like Pixar.

Certainly not at the resolution for a (feature-length) film like Pixar makes. They will still use large computing farms to get the job done in a reasonable amount of time.

AI can possibly seem to violate Moore's law, but it won't violate the laws of physics. If we are doing 1/3 of what our CPUs can do, then AI will make it so we can do the other 2/3 as well. And we can be really amazed at the results.

But also with "enhanced computing", things that once seemed incredibly difficult to program are now at least possible. Not necessarily "easy," but what once would take a team of 25-50 people (at least) can now be done sometimes with a team of 3. That's not a general rule, just that some things that once took many people can now be done with few, and faster than when it took more people.

Wizard-like stuff that once took a team can now be done by individuals. So the term "enhanced computing" is both telling and probably accurate.

And if you want, you can say that what computers could do already 10, 20 years ago is almost like magic. We know better, but for me it still feels a little bit like magic.

If you think of Harry Potter-- and I'm a fan of those books and movies, Ollivander said of Harry's nemesis: "He too did great things. Terrible, yes-- but great." It wasn't a compliment, it was an accurate measure. Of course for a young boy who just learned he was a wizard, it's creepy enough.

AI will do great things. Some of them will be terrible-- but great. And hopefully more of them will be Harry-like than Voldemort-like. But really, it will be both. AI is already used to help kill people. And I don't know how much we are ready for it. We should be cautious, and know that the best rules we come up with (like no doing magic outside the school grounds) won't be followed all the time.

No "Ministry of Artificial Intelligence" is going to be free of corruption or poor decisions-- nor would it be enough to stop all bad things that are done with or without approval. Either way, AI is here.

Perhaps the biggest difference between AI and human thought is the superficiality and bias. Humans have that sometimes, in very stupid ways, but we are more flexible. AI can magnify our stupidity-- think of the old adage about "knowing just enough to be dangerous." That's AI.

I'm not so much talking about "What happens if we copy people into AI versions of those people?" I'm really talking about the potential to try to make AI do what we think we want-- and getting far worse versions on average.

Because that's going to be very common, even humans have done this now and again throughout history. AI will lead us to a greater capacity for such mistakes. Just as AI can solve things that would take 100 people to solve, it can make mistakes that would take 1000 people to create.

At least with laws, there's a judge and jury as long as it's not artificial. We are certainly building corporations that have more power than a judge and jury do. But AI could do that too. My feeling about AI politically is still that it lends itself to many things, but it lends itself best (or at least most easily) to fascism.

That could be post hoc-- it's corporations and governments that are the most interested in it, so I could be describing what it lends itself to most easily by extrapolating it from the product of governments and corporations working on it. Still-- what are are developing now is like that.

People are trying to think of whether AI will be more good or more bad, and I'm not arguing for a neutral stance on it. I would say that if you look at all that computers have done both for our lives, and also to our lives, computing that is suddenly enhanced in ways that at least seem to go beyond the reach of Moore's law is exciting, but also justifiably scary.

What AI does is pattern recognition, and it can also impose patterns. This is said broadly because that's the broadness of the application-- you can find patterns similar to the way a person would, you can impose patterns similar to the way an artist would. Computers can do that without AI, but not at the same level as a person.

Today, we are designing software that can do those things faster and more tirelessly than people-- with similar (or sometimes superior) skill. Manipulating video, audio, tactile environments-- targeting, surveillance-- these are being expanded and developed all the time, not just in the future.

```
change_what = "new bicycles"
change_to = "baby kittens"
phrase = lineinput # the input command
p = phrase | split p, change_what | join p, change_to
now = p | print # the output command
```

the only parts you need to type to make it work are:

```
changewhat "new bicycles"
changeto "baby kittens"
phrase lineinput
p phrase split p changewhat join p changeto
now p print
```

The only syntax fig requires is # hashes-for-comments and **"**quotes for strings.**"** Bold text is meaningless, underscores are just part of our variable names. The rest is to group code visually. Let's remove comments and number each line:

```
 1 change_what = "new bicycles"
 2 change_to = "baby kittens"
 3 phrase = lineinput
 4 p = phrase | split p, change_what | join p, change_to
 5 now = p | print
```

Lines 1 and 2 just set variables **change_what** and **change_to** to the string to look for, and the string to change it to. The names are not special-- we could have called these **f** and **z**.

```
 3 phrase = lineinput
 4 p = phrase | split p, change_what | join p, change_to
 5 now = p | print
```

Line 3 is the first one that contains a fig command. **lineinput** is one of fig's 90-something commands with a specific name, and what it does is get a single line of keyboard input. And it sets the variable to that input. So when we say:

phrase = lineinput

What we mean is:

"Create a variable named '**phrase**', but instead of setting it to a number or a character string, set it to whatever characters are typed in at the keyboard." Right? That's easy.

```
 3 phrase = lineinput
 4 p = phrase | split p, change_what | join p, change_to
 5 now = p | print
```

Line 4 does a lot... it copies **phrase** (what we typed) to variable **p**, it splits **p** into parts wherever **change_what** is found, and it joins those parts using **change_to**:

change_what is "new bicycles"
change_to is "baby kittens"