Packages - Packaging Request #3353

Add back ghidra (was: [ghidra] Various freedom and technical issues)

2022-09-27 01:47 AM - gap

Status:	open	**% Done:**	0%
Priority:	bug		
Assignee:			
Category:			

Description

See PKGBUILD: https://github.com/archlinux/svntogit-community/blob/packages/ghidra/trunk/PKGBUILD

Freedom issues:
- Line 46:

    Ignore lack of licensing for YAJSW zip, packed FID datasets, and the native binaries

Technical issues:
- Not built from source
- Built from VCS, not sourceball
- Missing checksums

Yet more yucky Java

History

#1 - 2022-11-09 01:03 AM - GNUtoo

- Status changed from unconfirmed to confirmed

Thanks a lot for the information.

Ghidra is a java application and Arch Linux usually doesn't build java packages from sources.

I'll remove it for now and I'll then move this bugreport as a request for packaging ghidra properly.

There is also a lot of work needed for properly packaging Java applications in Parabola. We still have log4j if I recall well.

#2 - 2022-11-09 01:11 AM - GNUtoo

- Status changed from confirmed to open

- Subject changed from [ghidra] Various freedom and technical issues to Add back ghidra (was: [ghidra] Various freedom and technical issues)

- Tracker changed from Freedom Issue to Packaging Request

ghidra is now blacklisted.

So to add it back, someone needs to send a PKGBUILD (or a patch that adds that PKGBUILD) to build it from source.

It will probably require packaging its dependencies as well.

Denis.

#3 - 2022-12-09 09:38 AM - bill-auger

if there will be no libre replacement, this needs to be documented better at least - the blacklist entry does not include a BR reference, only "[technical] not built from source"; but that reason could not answer the most common question "why is this 'foo' not in the repos?" - "impossible to build from source" would be understandable; but im not so comfortable with "dunno if its possible - we didnt bother trying" - i could also accept "its undesirable java junk - we dont want it"

so, is there evidence of non-free or missing sources?
maybe someone need to try re-building it to find out?

#4 - 2022-12-09 02:50 PM - GNUtoo

bill-auger wrote:

> if there will be no libre replacement, this needs to be documented better at least - the blacklist entry does not include a BR reference,

I'll fix that

> but that reason could not answer the most common question "why is this 'foo' not in the repos?" - "impossible to build from source" would be understandable; but im not so comfortable with "dunno if its possible - we didnt bother trying" - i could also accept "its undesirable java junk - we dont want it"

Here the real reason is something like that: "We didn't have the time to try because we lack the time. Everyone is welcome to try (this is why there is this bug report), please send a patch and end up joining the Parabola project and help us fix the many bugs open (so if enough people do that, we might end up actually having the time to make libre replacements instead of plainly removing packages)."

Right now there is just too much problematic software and we need to find a way to handle that.

Given that we really don't have the time to add a libre version (I guess you will not spend time on that given how much work you have already with Parabola, and personally I've also a lot to do already, so if I did that I'd need to work full time on Parabola without being paid and I'll end up dead or in jail because squatting just became illegal in France and if you end up on the streets your life expectancy drops to few years).

And keeping that software in Parabola as-is without even warning users is not an option as we have a policy to always build from source, so users expect the software to be built from source. So if we hide the issue we would be misleading users.

So this brings the question of what to do. We could abandon that policy of always building from source and inform users about that. We could also make an announcement that gives users 3 choices:

- Either enough people join Parabola and bring back removed software, so the problem goes away
- Either we stop having the policy of requiring to build packages from source
- Either we drop packages not built from sources that we don't have the time to look into

As with "undesirable java junk", it is very subjective. Users and developers would probably not understand what it means (I don't for instance).

As I understand ghidra is useful and there is nothing wrong with the Java programming language per se (the issue are more in the build system used, or the way certain programs are written).

However given that we already have at least 1 big security vulnerability in Java (log4j) and that it's still not fixed, we could decide to drop Java completely because the work to update the Java packages we have is quite big (this is because upstream switched to build system we can't use) and that we don't have the time to fix it.

What do you think about these proposals?

#5 - 2022-12-09 02:58 PM - GNUtoo

Another idea would be that since we have a policy to build from source, we would keep only a stripped down minimal version of Java that we can maintain and remove all other Java software. People would then need to properly package the Java software they want/need in Parabola (or use other ways to install it).

That would probably be best.

#6 - 2022-12-09 03:41 PM - GNUtoo

Another option would be to remove all java packages from the database (with db-remove) and move all the java pkgbuilds in an experimental [java] repository.

For the JVM/jdk we could keep them if other non-java package depend on them or because it depends on itself. Though if no other packages depend on it, there is still the option to bootstrap it from jamvm-1 like Guix did[1] in case someone wants to bring it back.

[1] https://libreplanet.org/wiki/Group:Software/research/ProgrammingLanguages

#7 - 2022-12-09 07:39 PM - gap

We should identify and tackle the root cause of these issues.
AFAICT the shoddy packaging is a result of poor Arch packaging standards and the typical way software written in Java is developed and packaged.
Perhaps we could contact Arch and improve the packaging standards, and contact projects using Java to make them aware of these issues and recommend they develop in such a way that makes packaging and license checking simpler and easier.

#8 - 2022-12-11 12:19 AM - GNUtoo

gap wrote:

> We should identify and tackle the root cause of these issues.
> AFAICT the shoddy packaging is a result of poor Arch packaging standards and the typical way software written in Java is developed and packaged.
> Perhaps we could contact Arch and improve the packaging standards, and contact projects using Java to make them aware of these issues and recommend they develop in such a way that makes packaging and license checking simpler and easier.

I'm not sure how to do that or what is the root cause here.

My current understanding is that there is a big culture change going in the computer industry, and that it affects free software as well.

One one hand we have new tradeoffs that are made that on one side improve productivity but on the other, it will compromise on things like dependency control, efficiency and so on.

For instance if we take software written in C, there are known issues with memory safety, so if the code is not carefully reviewed and automatically tested, it might contains exploitable bugs that can lead to remote code exploitation. The main issue here is probably the time it takes to make the code work safely (like have 100% test coverage, etc).

With C, the infrastructure behind it is really good. For instance, in practice you have real control over dependency management, shared libraries enable to reduce resources consumption globally, some build systems like autotools support cross compilation, it works on micro-controllers too, etc.

If you take Rust, go, nodeJS, etc, different compromises are being made. If we take NodeJS programs, the amount of dependencies is too complex to handle for human beings. In many case that can give the control of your computer to any person maintaining any of the dependencies.

Here a way to fix it would be to use distributions like Parabola or Guix as they build from source and they are supposed to check checksums or signatures of upstream packages to make sure that nothing changed.

And here there is also a culture change in the opposite direction as there are efforts too to go in the right direction with reproducible builds or bootstrapable builds (like with Guix that works toward bootstraping its compilers).

The issue we have here is that the culture I was trying to describe before made it in various programming language. For instance if we take matterbridge (a program written in go), if we wanted to package it properly in Guix, we'd have to add more than 500 dependencies. And we cannot use some of the dependencies from guix and some other that are bundled in matterbridge: we have to either package all the dependencies or use all the ones bundled in matterbridge (about 500).

That mindset also got in the Java world. Before Java was more like C: we had build systems like ant. To build a software, after deleting the bundled libraries (in the form of jar files) we ran ant command (instead of make commands) and it built the software. And we usually had 1 package per dependency.

But now users are expected to build software outside of distributions with build systems like maven (for java), cargo (For Rust), npm (for javascript), the go build system, etc.

This makes development easy as it works on all operating systems easily, but it makes packaging super hard. Guix can easily create very lightweight containers for building software, so it can bring the same easiness, but that doesn't work well with software that has too much dependencies to be properly packaged.

The issue is that we still have to deal with programming language build systems in distributions (even in Guix) if we want to package software written in these programming languages.

So Java now uses maven (a new build system) instead of ant. In practice maven fetch unknown binary dependencies from "maven central" and we don't know where to find corresponding source code.

So the fix we have in Parabola for that is go straight to the source code release and in the PKGBUILD, run the javac commands by hand.

The slf4j PKGBUILD is a very good example of that. Here the main difficulty is to find the build order. And you also still have a lot of dependencies so both combined makes progress very slow.

As for Guix, last time I checked it had less Java packages than Parabola and all the packages it had was old because they stopped updating them when the packages switched to maven. There was some effort to bootstrap maven and try to use it though I've no idea how far it got.

In any case it's a good idea to watch what Guix is doing since we could do something similar in Parabola.

And using Parabola packages instead of installing them with Guix has a lot of advantages: Parabola takes way less space, its package management is more simple, we can mirror all its packages and source code (it takes about 200GiB), while that's not possible with Guix, etc. So just using Guix is not a drop-in replacement for packaging things in Parabola. In addition there is still a lot of things missing from Guix (like KDE, many programming languages like ada, C#, etc). Guix has other advantages so Parabola can't replace Guix either.

Though it might still be possible to get funding (for instance through NLnet) to improve java in Guix and backport that in Parabola (and add both in the application using the rationale that this work in Guix can also benefit other distributions, and that reusing it in Parabola (a distribution that might want these changes) is a way to make sure that it is reusable).

Personally I've already applied to get funding for improving the collaboration between various Android distributions (including Replicant), and I'm not sure to have the skills needed here (I'd have to look in more details to see if it requires to write compilers, how hard it is, etc), so right now I'm probably not the best person to tackle this issue more globally.

Though the page [1] I mentioned before is also an attempt to counter that mindset of not caring about things like good dependency management, compiler bootstrapping, etc, though it doesn't talk (yet) about build systems.

And it also shows that (at least some) of the issues can be fixed. For instance GCC now supports go, so you can at least have a subset of the same infrastructure used for C, so it might be good enough for some use cases (like you could probably cross compile go, use Makefiles, combine go and C/C++ easily, etc). Though I've not yet managed to use go with autotools but if it works somehow that would gives many of the same features we

relied on.

[1] https://libreplanet.org/wiki/Group:Software/research/ProgrammingLanguages

edit1: cross compile go -> probably cross compile go (I didn't check if it worked)