Reproducible Builds Summit 2024

September 17 – 19, 2024 Hamburg, Germany

Event Documentation

Aspiration

Phone: (415) 839-6456 • info@aspirationtech.org • aspirationtech.org

Table of Contents

Event and Agenda Overview

The 2024 Reproducible Builds Summit took place 17-19 September 2024 in Hamburg, Germany, followed by additional hack days.

The event was hosted at dock europe e.V. , https://dock-europe.net/

Complete details and event documentation are available at

https://reproducible-builds.org/events/hamburg2024/

The agenda was designed as a combination of planned sessions and participant-driven discussions, and specific topics placed into time slots based on pre-event input from those who will be in attendance as well as feedback during the event.

The following are some general guidelines regarding how the workshop was run:

- The agenda will be flexible and dynamic. Sessions will be adjusted and added based on suggestions and requests by both participants and organisers. Stated session and break times are approximate and may shift. But we plan to start each day on time and end on time or early, and we shall not be late to lunch :)
- Session topics and outcomes have been sourced from pre-event input from participants. We *welcome and encourage* additional session requests and proposals, as long as they focus on collaboration, co-creation and sharing of experiences, goals or visions, rather than being lecture-oriented or slideware-based.
- Full participation in the program is requested; part-time participation will work against the overall event goals. In particular, we ask all participants to refrain from scheduling side meetings during stated agenda times.
- This is a "devices at ease" event: use of laptops and cellphones during sessions is strongly discouraged with the exception of designated note takers. We invite participants to use the morning and afternoon breaks, as well as the lunch hour, to check in with external realities.
- We will follow Aspiration’s Participation Guidelines (https://facilitation.aspirationtech.org/index.php?title=Participants:Guidelines) and Event Expectations (https://aspirationtech.org/events/expectations) during the meeting.
- Overall, sessions are intended to be highly interactive. Facilitators' primary goal will be to enable collaborative and co-created outputs, constructively address questions, and support peer sharing. We invite all participants to bring your ideas, your questions, and your knowledge to contribute.

Reproducible Builds Summit 2024 - Agenda

Day 1

Tuesday, 17 September 2024

Opening session

The 2024 Reproducible Builds Summit will be called to order with a fast-paced kickoff that includes words of welcome from the organizers, brief participant introductions, along with overviews of the agenda, participation guidelines and meeting logistics.

Getting to know each other

Each participant was invited to talk with another participant they had not met yet.

Interactive Project Updates I

Several facilitators hosted short discussions, sharing updates about different projects and topics.

- Maven and Java ecosystem
- whatsrc
- binsider
- reproducible nikita / prefix
- Debian status

Break

Interactive Project Updates II

Several facilitators hosted short discussions, sharing updates about different projects and topics.

- openSUSE
- Tor Browser
- OSS Rebuild
- Aroma
- System Transparency
- Reproducible development environments at work... Made easy

Break

Mapping the Big Picture

Building on the mappings we did at the 2023 Reproducible Builds Summit, the group took stock of where things stand for Reproducible Builds across a range of context, as of the Summit. We also identified success stories, exemplars and case studies to be celebrated and amplified, while also mapping challenges, needs and unsolved problems.

Topics, issues and ideas that surfaced during this session informed how we structured the rest of the agenda. There were 3 primary brainstorms:

- Everything we need to talk about (the "Agenda map")
- Success Stories and Unsolved Problems
- Missing maps/lists/documentation/visualizations

Lunch

Strategic working sessions I

There were 5 parallel topics in the first round of working sessions:

- Getting started with reproducible investigations
- Making the business case for reproducible builds
- Rebuilder information exchange
- Debian supply chain
- RB problem prioritization

In this and all subsequent working session slots

Closing session

The meeting day ended with an invitation to all participants to share their goals and wishes for the following days, and words of gratitude for a productive start of the Summit.

Adjourn

Hack time (No notes taken)

Day 2

Wednesday, 18 September 2024

Opening session

The second day of the Summit opened with a summary of the work done during the previous day, and a brief overview of the plan for the day ahead.

Strategic working sessions II

- Rebuilders II
- Kernel reproducibility
- Getting started with reproducible investigations II
- Making the business case for reproducible builds II

Skill share sessions

Lunch

Strategic working sessions III

- Rebuilders III
- Kernel reproducibility II
- Getting started with reproducible investigations III
- Making the business case for reproducible builds III

Closing plenary

The day was brought to a close with the invitation to share proposals for the next day's working sessions, and the invitation to use the meeting space to hack together during the rest of the afternoon.

Adjourn

Hack time

Day 3

Thursday, 19 September 2024

Opening session

The final day of the Summit opened with the encouragement to focus the last day of the meeting on drafting action plans to be carried out during the following weeks and months.

Strategic working sessions IV

- Rebuilders IV
- Getting Started with Reproducible Builds IV
- Making the business case for reproducible builds IV
- Teaching RB at universities

Lunch

Mapping next conversations and next steps

The group used the final session to identify areas for ongoing collaboration. A focal point in this discussion was a visioning exercise enumerating 3 lists we will review at the next Reproducible Builds Summit. The prompt began “When we get to the 2025 Reproducible Builds Summit what MUST/SHOULD/COULD be true?” The idea was to enumerate baseline goals for the next year (“MUST”), additional high-priority goals (“SHOULD”), and stretch goals (“COULD”).

Closing plenary

The Summit ended with notes of appreciation for everyone who participated in the meeting and contributed to its success.

Adjourn

Hack time

Reproducible Builds Summit 2024 - Session notes

Interactive Project Updates I

Maven and Java ecosystem

- growing reproducible-central activity and contributions: https://github.com/jvm-repo-rebuild/reproducible-central/graphs/contributors
- output-file level GitHub badge, for example https://github.com/jvm-repo-rebuild/reproducible-central/blob/master/content/io/quarkus/README.md
- feedback: interest in other repositories than Maven Central: Android, Kotlin/Jetbrains

whatsrc

- https://whatsrc.org

binsider

- https://binsider.dev/
- https://github.com/orhun/binsider

reproducible nikita / prefix

-

Debian status

- 97% reproducible in CI
- reproducible docker images
- reproducible live-images
- debian-policy: we are very close to enforce no regressions and no new packages which are unreproducible
- snapshot.debian.org got fixed (required for rebuilders)
- now: setting up rebuilders...

Interactive Project Updates II

Several facilitators hosted short discussions, sharing updates about different projects and topics.

openSUSE

- https://en.opensuse.org/openSUSE:Reproducible_Builds
- https://lists.opensuse.org/archives/list/factory@lists.opensuse.org/thread/2XMBAI6G4ZFAS7LFWM2XPYHZYMG5C2GF/

Tor Browser

- https://gitlab.torproject.org/tpo/applications/tor-browser-build/
    - We build a lot of stuff, see the projects directory. We have to interact with a lot of different build systems (Firefox's, CMake+Ninja, Go, Cargo, autotools...).
    - We we have a series of bash scripts that are customized in real time with a tool Nicolas wrote (RBM: https://gitlab.torproject.org/tpo/applications/rbm/)
- Firefox ESR 115 -> 128 migration
    - Linux + macOS: no reproducibility problems
    - Windows: problems with the Rust toolchain, due to the GCC/mingw toolchain we used to build Rust std. We migrated to the arch-pc-windows-gnullvm targets (https://doc.rust-lang.org/rustc/platform-support/pc-windows-gnullvm.html)
    - Android: strange regression on the Gradle open source license generator. The patch is easy, but we hope to fix it upstream

OSS Rebuild

- https://github.com/google/oss-rebuild

Aroma

- AROMA stands for Automatic Reproduction of Maven Artifacts. We investigated how far we can go with simple heuristics to help Reproducible Central.
- https://dl.acm.org/doi/pdf/10.1145/3643764

System Transparency

- System Transparency (ST) is a combination of technologies for running computer systems transparently.
    - A system is transparent if a user of such a system can verify what's running on it.
    - ST combines verified boot, measured boot, reproducible builds, transparency logging and remote attestation.
- https://www.system-transparency.org/

Reproducible development environments at work... Made easy

Mapping the Big Picture

Building on the mappings we did at the 2023 Reproducible Builds Summit, the group took stock of where things stand for Reproducible Builds across a range of context, as of the Summit. We also identified success stories, exemplars and case studies to be celebrated and amplified, while also mapping challenges, needs and unsolved problems.

Topics, issues and ideas that surfaced during this session informed how we structured the rest of the agenda. There were 3 primary brainstorms:

Everything we need to talk about (the “Agenda map”)

Tooling

- Creating underhanded RB contest
- Easy actionables to do in a team to facilitate R13Y
- Make reproducible builds by default in build tools
- Make repro builds default in compilers
- Bridging CI/CD with package builds
- Improve/add heuristics to reproduce Maven artifacts
- Produce an easy way to check reproducibility of signed binaries
- Automating reproducibility checks

Rebuilders

- Consensus implementation for rebuilders
- How to share rebuild results between rebuilders
- Rebuild definition formats
- Rebuilder network design
- Interchange format for rebuild info
- Do we worry about reproducible package version selection? (e.g., repro lockfile generation)
- Trust model for human contributed rebuild definitions
- Encouraging diversity of rebuilders

Languages

- Python R-B issues
- Rebuilding Python packages that are available on PyPI
- Lisp+Scheme R-B issues
- Rust R-B issues

Distros/OS

- How to make Debian-based docker images reproducible?
- macOS codesigning and reproducible builds
- Immutable operating systems are the solution?
- BOotstrapping reproducibility

Docs

- What are the milestones for this year?
- Mentor programs like GSoC/Outreachy?
- How to contribute to the cause?
- Carrot/sticks to get pkg maintainers to be reproducible
- Trigger end user interest

Source code

- Establishing canonical source repos
- Source tarball reproducibility
- How to avoid another “snapshot is broken” situation

(no category)

- Exgend R-B to quantum computers

(no category)

- The use of transparency logs

Success Stories and Unsolved Problems

Success stories

- Bernhard’s RB-OS ring 0 reproducible
- Independent rebuild check is part of release workflow of Apache Airflow (data science)
- R-B mentioned in SLSA 4
- Doing the 8th R-B summit
- Timestamp issues solved by SOURCE_DATE_EPOCH
- A network of rebuilders exists
- Independent Arch Linux Rebuilderd in an Applied University :)
- Arch Linux has independent rebuilders for real-world binaries
- Projects are happy to take patches to ensure reproducibility (in my experience)
- Practice of R-B is known and accepted by many developers
- [Meta] R-B website getting an update about success stories
- Debian containers are reproducible
- Conda-packages are reproducible using rattler-build
- apt.vulns.xyz documents how to do reproducible 3rd party apt repos
- repro-env tracks “traditional” Linux build environments
- Using reproducible development env. is an amazing experience
- apt-swarm implements an authority-less p2p transparency log
- RB + SBOM permitted to find broken dependencies in releases
- Finding bugs: libxslt issue 37 “puzzled why it took so long to discover this issue”

Unsolved problems

- Establish canonical source repos
- Agreeing on source code consensus
- How to systematically detect toolchain reproducibility regressions
- Reproducible day to day dev builds
- Document format and protocols for rebuilder network(s) missing
- How do we build a system of attesters for proving reproducibility?
- Still not a good enough final user (regular, simple human) motivation and publicity
- Motivate Maven devs to add timestamp to their pom files
- How to make the world benefit from R-B
- Filesystem/VM image reproducibility
- Awareness in IT, crypto, and cybersecurity fields
- Linux secure boot and reproducible builds are incompatible
- Deriving build instructions
- No contact with proprietary tool vendors (e.g., Apple)
- How do we create fully reproducible infrastructure? Is Terraform enough?
- Reproducibility requirements in cryptographic standards (e.g., NIST, BSI, …)
- Embedded signatures making build non-reproducible
- Many open source devs I talked with don’t know about reproducible builds (but they agree it’s a good idea once you explain it to them)
- I can’t find a way to sell r13y to people that are not aware of it :-(
- How a maintainer can declare/communicate intended non-reproducible parts of binaries
- We need a serious marketing effort for R B adoption
- People afraid of learning new tools/tech
- Transparency logs, how to do them, how to use them
- Haskell’s GHC has non-deterministic output with concurrency enabled
- Solve RB for iOS ecosystem (Apple modifies .ipa uploads)

Missing maps/lists/documentation/visualizations

- what is blocking progress
- open issues by programming language
- A non-distro/all-distry list of unreproducible software
- RB issues with doxygen: common problems&solutions
- I want a list of all pypi pythron reproducible packages
- CVEs but for reproducibility
- List Security bugs in RB processes and tools (diffoscope, verifification scripts, etc)
- list services and/or infrastructure we depend on
- different types of attestations, pros and cons of each of them
- homepage/doc: other classes of repo issues besides what is already documented?
- to have some guidelines about how to be a reproducile build tools
- review docs website for direct links to targeted actionable headings
- cyber resiliance act requires SBOM, what are tbe best practices to implement them?
- best practices to investigate and resolve reproducibility bugs
- how to properly introduce reproducibility in a team? why would we do that?
- doc explaining how to set up and run rebuilders
- easy TODOs for student projects (3-6 month wokring 3-4h/week)
- format to communicate rebuilder capabilities
- “service [name] only has reproducible deps” green checkmarks
- rewrite strip-nondeterminism in
- tools to help building reproducibly
- divergence between differnt packaging reproducibility for same upstreams
- what is the standard way to make go binaries reproducible
- how to properly sign APKs to then run APKSIG copier
- how can I gather gradle dependencies in the proper way
- source packages to upstream repositories

Strategic working sessions I

Sessions at the 2024 R-B Summit took on a new and different form than past summits. Many sessions continued across multiple session slots, in some cases up to 5 sessions. Hence some of the notes that follow are aggregated across all those sessions, and thus do not map 1-1 to the sessions listed in the agenda above.

Getting started with reproducible investigations

Getting started with reproducible investigations; session I

Upstream: https://salsa.debian.org/reproducible-builds/reproducible-website/-/issues/56

Objective

How can we make it easier to write software that is reproducible from day zero?

Benefits

Our audience is developers learning any software language/ecosystem. If we can reduce the number of build reproducibility problems that are introduced at an early stage, before that software is packaged by distros, then everyone's job becomes easier.

- No need for faketime or similar stuff, when you don't have timestamps in the first place

- They can provide configuration for the build systems, etc...

Design considerations

Realtime feedback about this during development - perhaps between commits - either from continuous integration or using on-demand repro-build-test could be useful.

Instead of building a complete package and comparing the output, an incremental approach that compares the intermediate artifacts produced at each step of the build could provide a generic solution.

A problem for reproducibility: storage is more expensive than CPU time.

Being able to check every intermediate artifact would require a lot of storage.

A logically simple, although sometimes imprecise, algorithm to report the earliest appearance of nondeterminism could be to compare the two output directories and to report the file with the earliest created/modified timestamp as the first problem occurrence.

Is this a good target to address though?

Reproducibility might not be an early problem, and might appear later.

Although that is true, developers often begin writing software on their own individual code branches, meaning that external changes that introduce regressions should be unlikely. Providing information when reproducibility problems begin to occur, and information about common mistakes that can cause them, could be valuable during that learning phase.

Should we go back to think about common errors, and e.g., add more documentation pages.

Our goal is to direct users to the good steps to get reproducible builds.

The Commandments of Reproducible Builds (https://reproducible-builds.org/docs/commandments/) could provide reference points.

How to deal with reproducibility problems induced by dependencies?

Are we missing tools to investigate them?

Other causes: time bombs, certificates that expire and are embedded in the software.

A good idea is also to compare reproducibility of a certain package in various distribution.

Some distribution might make some normalization that makes that package reproducible.

https://ismypackagereproducibleyet.org allows to do so.

Is building twice in a row on the same machine a good repro test?

It's a very basic test. If you fail it, it means you won't be reproducible for sure :)

But it won't catch problems such as CPU dependency (e.g., number of cores), or maybe dependency on the date (rather than the full timestamp), paths (unless you change also that).

However, definitely worth to have these kinds of basic checks in a build system.

To sum up: we've drafted some ideas for tools that could run both locally or remotely to check a basic threshold for reproducibility.

Making the business case for reproducible builds

- Prioritize two industries first:
    - Cryptocurrencies/blockchain and finance
    - Cybersecurity
- Main question: how?
    1. "Evangelizing": spread the word about R-B
    2. Better tools: novices should be able to use them
    3. Alignment with compliance (e.g., rules or requirements that require R-B)
    4. Sell R-B as a good technique/tools for developers, so there is a R-B culture at companies
    5. Help detect security issues (e.g. using diffoscope when there is a "weird" change)
    6. Start this at universities: spread the word to stduents
- How would we spread R-B at universities?
    - Talk about this in conferences
    - University hackathons: e.g., in any generic hackathon, at the beginning explain the concept of R-B, the tooling, etc. At the end, perhaps only accept solutions that are reproducible. Or, alternatively, give an special prize to R-B solutions.
    - Explain students how to produce reproducible documents (e.g., with LaTeX)
    - Perhaps introduce R-B into the syllabus of basic lectures in related degrees (e.g., Computer Science, Cybersecurity, etc.)
    - If certain key universities adopt R-B to their syllabus, many other universities will copy it. For example, if CS50 in Harvard talks about R-B, this will reach millions of students
- It's a tricky quesiton how to incentivate/motivate open source developers about R-B. That is why we should align R-B with the parties benefiting the most out of it. For example, we should focus on companies providing some software with some compliance rules, etc. instead of the open source developer.
- The best way to spread the use of R-B would be to ask goverments or certain industries to enforce R-B for their software
- Maybe we should start lobbying for R-B
- Maybe for (non open-source) companies, R-B can still be useful. For example, for companies working on machine learning or AI
- Ideally we would have a good project, that can generate money and invest part of that money for lobbying for R-B efforts (e.g., like Coinbase does). This probably requires a good idea and luck

- Perhaps, a more practical (and less risky) approach that requires less money (but more passion and time) would be to start the evangelization in some smaller sector like targetting the Python ecosystem, and help them make it reproducible

- Key takeway: we should align R-B with where the money is

- CI is not enough anymore. R-B can be a solution for a large developing problem. But this will only work if there are tools available that are simple and easy to use, to they don't impose any extra work on developers

- Companies that want to protect their IP but also provide their users some kind of R-B logs from a verifier without providing the open source

- Another target for the evangelization effort: Security auditors. They should understand R-B, their benefits, etc.

- Another idea: lobbying about R-B requirements for NIST standardization efforts (e.g. for the recent PQC effort)

- We should do a R-B course

- Maybe we should modify the R-B website so that it convince people about the need for R-B: we need case studies or success stories

Rebuilder information exchange

Four different topics:

1. Consensus
2. Capabilities
3. Vulnerabilities
4. Build definitions

What is a rebuilder?

A service that does rebuilds, a person who does it, a hosted platform. Examples:

- rebuilderd
- reproducible central
- oss rebuild

What is the problem we're trying to solve?

Some definitions:

- Rebuilder takes a recipe for an artifact and executes it to create a reproduced artifact.
- Rebuilder network: many separate entities that operate rebuilders trying to builder
- A builder just executes a recipe. A rebuilder is different because it includes an expectation

If we don't have an authoritative binary (not central repo) we need to build consensus on what's "right".

An interesting question is whether we use the authoritative artifact to derive the recipe, or whether the recipe can be defined in a vacuum.

OSS Rebuild and Reproducible central are trying to matches upstream. Tor is trying to build consensus by building honestly.

Is the output just "yes or no". We think the output should also include the recipe. It helps people understand how the rebuild was executed. The recipe is not unique, the rebuild recipe might pin versions, the recipe might include extra components that are not strictly necessary to reproduce the artifact. Because of that there are many recipes for a given artifact rebuild.

Two different goals. OSS Rebuild and Maven central are trying to root out the actual recipe. Tor is building the same thing many times to build consensus.

How much should we try to emulate the same thing? How much should we try to add variance as a way of increasing the variety as a way to increase trust?

Should we have a standard output with:

- A description of the builder (build type)
- The target we're trying to build (ecosystem, package name, version)
- The build steps required (recipe)
- The output that was generated (maybe both pre and post normalization hashes)

One party can normalize the upstream. Another party can do a rebuild with normalization. They should output the same normalized hash.

What properties do we want from a build definition?

It should be zero parameters and easy to rebuild.

Ideally we'd like the build definition to be hermetic.

It's possible that there are two definitions. One that's non-hermetic and fetches things dynamically. And another that's an encapsulation that makes it hermetic (cached network requests, pinned versions with a lock file, etc).

There is room for the final recipe to be "compiled" and not human readable. Something like a docker image. It's not ideal

A standardized recipe format is almost impossible, but we can define some useful properties. On nix you can use the commit hash to define the build env. Alternative is the BUILD info file which captures the environment.

What outputs should the rebuilder have?

- Boolean success/failure
- hash of the output artifact
- hash of the source code

There are different users:

- Rebuilders who also want to do the rebuild
- Security engineers want to inspect the build system used
- Users just want to know the thing is secure

Motivating considerations for rebuild formats:

- For rebuilderd today, the recipe is not required because the upstream package includes the build instructions already
- A build definition by itself does not include a package identifier
    - For linux distributions this is awkard because everything is keyed based on the named identifier
    - For rebuild definitions broadly, it's helpful for the rebuilder to include their intent with the rebuild attestation. Just building something from source is not the same as claiming this source produces this named package.
- For Tor, there is no known hash when the builders start. This is because the builders are trying to create a consensus.
- For android APKs there is a signature on the upstream. It cannot be reproduced in the rebuilder. One solution is to provide the signature as an input on android apk rebuilds and the rebuilder can just use it. Another solution would be to strip the signature off before doing the comparison.

Rebuild findings signals:

- Garbage files (.DS_STORE)
- File path exists in the repo but the file contents do not match
- Windows line endings
- Git tag doesn't correspond to the commit that does rebuild
- Optionally a human notes section
- A human populated enum for root cause (Our rebuilder needs to be improved, a shared build tool needs to be improved, package owner needs to make a change in their package)
- LLM notes
- Github issue
- Reproducibility issue OSV
- Related: https://salsa.debian.org/reproducible-builds/reproducible-notes#packagesyml

Debian supply chain

(The order of notes, does not necessarily
 reflect the order in which they were discussed)

- Document quirks
  * packages on snapshot.d.o not signed
    Release file is signed, but key is expired
  * binNMU
    need to take old sourcecode + buildinfo file
  * binNMUs
    can lead to different files with the same name
    with different contents
    (binNMU for different distribution/suite)
  * buildinfo files not published by debian
    (but instead by buildinfo.d.n)
  * buildinfo.d.n vs buildinfos.d.n
    buildinfos.d.n has buildinfo for binary package uploads
    (e.g. when uploading to NEW)
  * version numbers in buildinfo files
  * source upload: developer signs source packages,
    but signatures are thrown away
  * dpkg-genbuildinfo generates buildinfo file
  * debrebuild only respects the packages listed
    in buildinfo
  * buildinfo does not document /ALL/ installed packages
    e.g. fakeroot not listed in buildinfo
    problem: e.g. installing qemu can change
    the behaviour of a package build
    do we need to list /ALL/ installed packages?
  * fakeroot solvable:
    builder should respect Rules-Requires-Root

  (and rebuilders could pick that up as well)
  however: right version of fakeroot?!
* should buildinfo include the builder used?
* everything that ends up in testing
  was built using sbuild
* life of a debian package
  (including infrastructure bits,
   there exist several talks @ DebConf
   both regarding package "lifecycle" as well as infra)
  also: security updates
  (built on embargoed machine: no (or delayed) build logs, buildinfo?!)
* include buildinfo in the archive
  debian bug: bugs.debian.org/763822
  dak does not know really know about buildinfo files
* buildin-pho
  has a database that can be queried
* convention for filenames of buildinfo
* ftpmaster are running their own sbuild
  diff is small
  builds are running stable
  own sbuild branch to be able to experiment
* packages that are needed for debuerotype
  are 100% reproducible
  (scripts that automate rebuilds on Ubuntu)
  - including parsing build logs for the presence of fakeroot)
* apt knows the hashes of downloaded .debs
  but throws it away after verification.
  apt /should/ record the hashes somewhere
  so that the buildinfo file could use those hashes
  (by doing the intermediate step of checking the versions
   and retrieving the corresponding hashes)

* alternative: instead of recording hashes in the buildinfo
  ensure/require that we have a 1:1 mapping
  between package_version_architecture to package hash
  (see binNMU problem above)
* lots of tools (especially dpkg)
  assume that package_version_architecture is
  a) a unique identifier
  b) there are no hash collisions
* merged /usr was (for the most time)
  a repro issue
* Ubuntu wants to provide reproducible builds
* launchpad provides buildinfo nowadays

RB problem prioritization

Collecting and Prioritizing Problems in r-b

One participant has a list of unreproducible openSUSE packages, and so does Debian, but there is more than that that is tracked nowhere.

Generic tasks that apparently nobody is working on:

- Apple IOS apps, (Situation is better with Google app-bundles because of OSS tooling)

Important issues on FOSS stuff that is shared across distributions:

- emacs, other scheme and LISP interpreters
- Haskell ghc has issues with parallelism
- octave
- rust packages
- PGO issues in gcc, python, etc...
- upstream existing distro fixed
- kernel module signing vs kpcyrd's proposal on ML: this requires somebody to interact with Linux upstream
- r-b tools development, e.g. reprotest
- make diffoscope smarter (more format-aware diffs: https://salsa.debian.org/reproducible-builds/diffoscope/-/issues/?label_name%5B%5D=comparators)
- use diffoscope as a library (https://salsa.debian.org/reproducible-builds/diffoscope/-/issues/68)
- libxslt (suse and debian have different patches, none upstreamed) https://gitlab.gnome.org/GNOME/libxslt/-/issues/123
- gstreamer-plugin-rs
- libcamera
- numpy

How to prioritize

- by impact (how many users are affected)

- by complexity / difficulty
- both combined

Todo:

- somebody copy the above list in the "how to contribute page"
- from there link to more detailed (bugs?) lists, like for Suse, Debian, ...

maybe: promote the list so people who would like to have help for something, they could add to the list? probably needs some rules so people don't just dump all bugs

-- Debian lists:

https://tests.reproducible-builds.org/debian/unstable/amd64/index_FTBR.html

https://tests.reproducible-builds.org/debian/unstable/amd64/index_no_notes.html

https://tests.reproducible-builds.org/debian/unstable/amd64/index_notes.html

Please refer to additional notes in “Appendix 1 – Additional notes from RB Problem Prioritization Session

Strategic working sessions II

Rebuilders II

Group A: transparency logs / outputs / consensus

A large group of people are interested in "rebuilders" -- and figuring out what that means.

We split into groups focusing on {output and how to compare it}

vs {recipies and rebuilders and diversity of inputs}.

This is the notes for the group focused on outputs.

- It would be good to have a rebuilder swarm -- collaborating diverse entities.
- It is a "policy" matter for what to _do_ with information about whether or not things successfully reproduce.
    - The rebuilder network does not decide this. Policies are things developed by other (potentially various) groups downstream for figuring out what to do based on the data produced by the rebuilder network's efforts.
    - Recommendation for policy design: think about where the offramp is if something _doesn't_ produce: Does it fall to end-user package installers to _abort_ and refuse to install unreproduced software? Or should the policy kick in at any earlier phase of rollout pipeline, and an organization should not _tag a release_ if the reproduction has not succeeded?
    - (No intention here today to build One True Policy framework -- just recommendations on considerations.)
- A tricky topic: reproduction of commercial or limited-license software.
    - (Not recursing into this!)
- What are the incentives of various folks to run rebuilders? Are resources just shared and contributed out of the goodness of heart? _Are_ rebuilder resource pools sharable?
- A property we want to work on in this group: How can we give people increased faith/confidence/audit/whathaveyou in software package reproducibility _when they **do not** run the rebuilder themselves_?
    - This is an important refinement to the goal because the rebuilder-i-run-myself is a trivial story for trust propagation, and at the same time very resource-intensive. So we want to look beyond this!
- We propose development of: A database where you can...
    - push a record of what you executed as a build, and what the result was.
    - can one push anything without authentication? hmm. maybe problematic.

- query by: git tag?
    - tag? or by hash? or both?
- want to see signatures on any result record that is pushed into this.
    - build service owner's signature?
    - how about build _machine_ signature?
    - both?
- *expect uptime*. You can't make a release if this is down, so uptime is critical.
- What kind of trust will become nested here?
    - And how to we make it less?
    - This should be decentralized!

- Would consumers of this database want to query filtered based on rebuilder signature keys? Or sheer numeric quorum?

    - Move that to a "policy" thing again -- people will want various definitions here. Both approaches may be valid.

- Recurring theme: "policy" system consumes from this database, and is entirely external to the database. There's room for more than one policy. _But no need for more than one database_, if we design it right.

- What if the "database" is a git repository?

    - Let's generalize that to any append-only log :)

- Would we expect many instances of such a database/log?

    - Well, probably.

    - There will be different opinions out there about admission criteria. (Spam will be a problem if not...)

    - Can there be aggregations if there's many of these?

    - Is it reasonable to consider individual build machines might have their own append-only log?

        - (if yes, then _very_ much need to cover the aggregation concept, clearly.)

    - Phrase "inclusion proof" comes up a lot.

    - Compare and contrast to Certificate Transparency logs -- many similarities, but it may also be even harder to decide what is admissible to a Build Transparency log, because it's more expensive to do a build, and whether the result is reproducible is begging the question. So admissibility of new entries has an almost inherent good-faith component.

        - -> very very likely going to result in plenty of different log instances, and each of them likely having different admissibility criteria.

- So let's just make sure this is okay, and the system offers value even in this case!

- What value is unlocked by having a shared log/database?
    - Monitoring!
    - Uptime!
    - Shared infra! (And logs are so cheap to run that this is actually reasonable to talk about offering as pure public service.)
    - Keeping each other honest about append-only!
- Can we have a shared agreement of schema of this database/log?
    - Can we have some useful shared schema for this _without getting stuck requiring one schema for build recipies_?
        - Concerned about this because getting stuck on a dependency on designing One True Recipe Schema sounds like getting stuck period.
- Interest in people even at the edges being able to specify their own "policy" -- for example identifying specific rebuilders that they trust.
    - -> Means that the log/database has to be inspectable by anyone.
- Interest in making sure "offline query" is possible.
    - This should be possible in anything based on a merkle structure.
- There is some value in consolidated (aggregated) logs...
    - Discoverability of data!
    - Arrow of time and append-only is better locked-open by having more eyes on it.
        - Making sure records can't disappear is an important part of being able to trust this system.
        - "Witness" is the name of something watching that the log goes only-forward.
            - Witness is not the same as Rebuilder! Different values provided -- also Witness is vastly cheaper to run, requires less protocol agreement, and are generally more unconditional and can usefully run on autopilot.
- (Conversation about enumerating the values of transparency logs, which we're concerned not everyone is familiar with -- but we'll hope that's ground we can find existing reusable materials to cover well enough, so we won't focus too much longer on that in this session.)

- More discussion about if we can get any more concrete about what might belong in the log, and what might someone want to query.
- The hash of the attestation record is a crap search key. You cannot discover it unless you already know the record.

- It's a great primary key! -- but it's a crap search index key.

- Debate about whether this can be addressed by existing infrastructure. For example existing package distribution channels could ship packages with metadata that points at a selected set of attestation records.

    - This might have slightly different centralization of trust.

- Only possible schema: an attestation record contains:

    - a "buildertype" -- freetext string, like a plugin name

        - (some kind of regisistry for this -- handwave. it's just supposed to be a disambiguation field that enables some kind of "plugin" vibe and extensibility, lets some of the other subsequent fields be a little opaque.)

    - "recipe_fingerprint" -- opaque, only expected to make sense per buildertype, no need for the log/database to understand this in any way.

        - should reference a build instruction / recipe that was used to do the build that this record is attesting to.

    - "result_fingerprint" -- _can we standardize this in any way?_

        - A query that is group_by result_fingerprint is very likely to be desirable! If the size of that result is greater than one, that means we're winning!

        - Would this be namespaced by builtertype?

            - **Hopefully not!** It would be desirable to be able to query if different ecosystems have produced convergent results.

        - (Look at whatsrc.org ! Can we have something like this... for hashes of built filesystems, in the same way whatsrc is hashes of source filesystems, and manages to be distro/ecosystem/tool agnostic)

    - signatures? (which? whose?) (could this have standardized format and meaning across all buildertype? maybe? unclear if requirement or just niceity.)

        - hash of the pubkey would also probably do just fine.

            - because someone walking this may prefer to just ask that pubkey's owner for more information such as a signed more complete reason.

    - would package names belong here? (there is much less way to "trust" these are correctly tagged...)

    - timestamp might go in here just because.

        - some dissent, because what are you going to use this for?

            - could use use a sequence number instead?

        - if this is included, would log/database maintainers verify this at the time of append? (sure hope so.)

- or is this when the submitter claims they did the rebuild? That's a very different thing... and cannot be verified, so that's spicy.

- Lessons from the past? Severe dissent among developers of previous log systems about whether

- _We should still assume external systems will point into the attestation record ID_, from their own various additional data. That's fine; that's great.

- Is there a value in being able to detect identical outputs... across buildertype/ecosystems?

- Exciting discussion.

- On the one hand: _maybe limited_. Today, tomorrow, and the next day: things will tend not to converge. So being able to ask if different ecosystems have produced the same output filesystem is "useless".

- (Dissent! At lunch, we find some people have observed getting identical dot-SO files even across distros! It's not common but it's possible and it's been seen in the wild!)

- (In particular, as long as things are wrapped in e.g. a deb file and an rpm file, it's a goto-not-equal immediately. But: maybe we could agree to publish hashes of unpacked filesystems, as well as a separate record for the fully packed result.)

- On the other hand: can we have a "north star" goal of being _able_ to talk about this kind of result equality? Yeah. It would be neat.

- Other values of recruiting and participating in a pool of resources:

- diverse hardware properties (things we'd usually rather ignore, "shouldn't" matter but maybe sometimes do, expensive to buy all this diversity yourself, etc).

- diverse jurisdiction.

- Important to think about how existing groups will adopt this and where it will fit into their flow.

- Spam is a problem in append-only logs and needs some early discussion.

- Redaction is also a major challenge and so needs early consideration.

- One possible address of this is having all fields of any log schema be hashes, which point to documents that are stored elsewhere.

- And fortunately, in practice, everything we've seriously considered during this discussion happens to fit fine within that.

Trying to distill a mission statement for this output log aggregation topic:

> **To make rebuilders useful, there needs to be a way to aggregate and share their results.**
>
> **... and make them usable and available to consumers of the ecosystem...**
>
> **...without introducing a new single point of trust.**

("consumes of the ecosystem": both: release managers; and also end users, independently, in an end-to-end way. (Release managers should be capable of aggregating trust (as they already do); pushing it to the edges is simultaneously desirable.))

Recurring observations (from this session, as well as from earlier years):

- transparency log yes
- "policy"

A few more things were discussed in lunch:

- One person asked about the draft schema above: "what about multiple outputs from one build?"
- Should we have in the log row schema... a list of results? A map of named results?
- Or should we simply suggest that multiple attestation records are submitted? Maybe with a receipt-fingerprint that just has a "_1", "_2" etc suffix.
- What different features and queries are made possible by those two different approaches?
- What differences in scalability? Are we talking about an average of two outputs or of hundreds (e.g. per-file granularity would be large numbers).
- If we try to reuse some existing transparency log projects... Do they have limitations on the complexity of log record schemas that we should be aware of and be constrained by?
- One person suggested that other transparency-log-like systems already exist in the wild for other purposes... such as anti-virus vendors maintaining large (public...?) databases of the checksums of dot-so/library files. This is used now for both detection of those files in the wild, but also for productive purposes like making a publicly discoverable link to debug sources. But

whatever their original intent, those records already exist -- can we align with them, and get a richer ecosystem as a result?

Group B: Rebuilder Recipies / inputs

* is/could rebuilderd be a common ground?
* What constraints are there?
* The builder defines what it takes as input for the build
* What should be allowed to be 'different' / encouraged diversity vs standardized
* What are the ecosystem-level flexibility
* script vs container vs vm
* adding structure to workers / interfaces could be useful
* currently captures a boolean of success and diffoscope on failure
* allowing a recipe input
* different workers within the same ecosystem may need an extension
* Rebuilder has integrations with upstream registries to get tasks
* Rebuilding as we discussed yesterday takes a recipe as an input argument, but in the current structure it's implicit / a conversation between the ecosystem-specific registry and worker

* it would be useful to have as an explicit artifact in output
* What do we get out of the recipes?
* recipe may be log of what has been done
* Is identification of the 'minimum needed constraints for stability' useful?
* Persona: knowing you've archied enough to rebuild
* Persona: being able to trace supply chain implications
* Persona: identifying implicit constraints that need to be explicit
* There are always more externalities - can a recipe be 'complete'?
* Do we accept we can never capture everything?
* How do you resolve rebuilder differences given uncaptured information?
* example: change in default umask

* It's hard to figure out what happened starting from an underspecified recipe than vice-versa

* A less specific *stable* recipe is 'better'

* Different workers will have different opinions on how 'complete' recipes will be

* Output of a specific rebuild worker should potentially have enough information to re-run

* Persona: consensus / security / stability

* There's a distinction between a 'recipe' and a 'rendered recipe' with what happened

* Persona: check build works over new version of source -> is this just CI?

* There used to be 'apache gump' that would identify potential downstream breakage, but it was very expensive. figuring out what is useful to test is hard

* The output interface needs an option for 'failure' in addition to artifact+log

* Challenges: supporting mac and windows / multi-platform

* If you look at layers and syscalls you should be able to figure out what subset of the image 'matters'

* Worker 'capabilities'

* Called 'backend' by rebuilderd

* subdelinieation within an ecosystem

* How much of this should be handled by rebuilderd vs is different instances?

Kernel reproducibility

No notes were captured for this session.

Getting started with reproducible investigations II

Are we sure the idea we came up with yesterday is a good one?

Sometimes intermediate results don't match but then the final result does.

Also, maybe we could suggest using reprobuild after all.

Q: Is setting up reprotest straightforward or is it difficult? Does it involve running VMs/containers/some other isolation or will it run on your system?

Probably something easy would be better to get people in, otherwise they might push the reproducibility aspect to later (unless it's straightforward to test).

Reprotest can use isolation, but it can also run on the system. Also, it's primarily aimed ad Debian packages, but it can actually run also other commands.

So, it should be easy enough.

Reprotest can vary a lot of settings, but some require higher privileges.

So, it can be a problem for running it in CI, for example.

When we check for reproducibility, there are two different kind of outputs:

1. What caused the changes? Reprotest can help with this
2. What are the changes? Diffoscope will help finding them out

Having a tool:

+ it can run very often, and detect a problem when it's introduced

- it will need to be integrated with a build system, so it will eventually need some manual work

Alternative: make documentation more interactive.

+ Thinking to documentation makes you focus the actual problem, without thinking about the technicalities that an automated tool might involve.

- If you want people to adopt something, you have to make it as easy as possible.

+ If you make documentation clear enough, you can then use it to start thinking about automation (e.g., if you elaborate documentation so much that you can come up with a checkbox list, then it'll be easier to automate it)

Possibility/suggestion: hire a technical writer to polish the documentation.

Documentation shouldn't be a wall of text, as people won't read it, or will be discouraged.

So, we should have a generic enough flow, with references to other small pages, or section that can be collapsed (and will be collapsed by default).

Question: do we want to tell users to use some compiler/build system/environment settings/arguments instead of their defaults, to exclude some problems in advance?

No, compilers will often have these settings by default. We can mention them during the investigation.

Also, the less time to get to an actual output will increase the morale of people.

While writing documentation, we might find some pain point (e.g., reprobuild is difficult to get to use - I'm inventing, not sure it's the case). So, we should try to list these pain points somewhere, to try to improve them.

It's not an explicit part of our problem, but if we find it, we should note it somewhere, in case we can back to it later (or somebody else is interested in taking it :)).

Document 1

Title: Reproducibility Quickstart Guide

This is a brief guide to help you get started writing software that builds **reproducibly**[link].

The easiest check that you can perform, without installing any additional software tooling, is to build your software twice and to compare the build output files.

[tip: comparison] A common approach is to [compare cryptographic hashes](https://reproducible-builds.org/docs/checksums/) rather than the artifacts, but using diff tools or the `cmp` command are also valid alternatives.

This works as long as the builds are reproducible byte-by-byte, but embedded signatures make this difficult. You can check [this page](https://reproducible-builds.org/docs/embedded-signatures/) for some suggestions on how to deal with them.

If the results differ, then you have found reproducibility bug either in your software or in your toolchain, and can proceed directly to the troubleshooting[link] guide.

If the output is identical, then you should add more variance to the build environment to examine less-obvious factors that might influence the output:

```
┌──────────────────────────────────────────────┐
│  Define what output needs to be reproducible │
└──────────────────────┬───────────────────────┘
                       │
          ┌────────────▼────────────┐
          │    Build your project   │
          └────────────┬────────────┘
                       │
                       ▼
          ┌─────────────────────────┐
   ┌─────►│      Build it again     │
   │      └────────────┬────────────┘
   │                   │
   │      ┌────────────▼────────────┐                 No
   │      │ Are the results identical? ├──────────►│ Troubleshoot[link] │
   │      └─────────────────────────┘
   │                   │ Yes
   │          ┌────────▼──────────┐
   └──────────│ Add variations[link] │
              └───────────────────┘
```

Document 2

Title: **How to add variations**

Rebuilding software on an individual machine and obtaining the same output does not guarantee that the software will always build reproducibly.

For example: a program that embeds the hostname of the build computer would not be bit-for-bit reproducible when built on systems with different hostnames.

Changing the compiler (and version) you use could also introduce differences. However, to achieve reproducible build results, it is generally acceptable to specify precise toolchain version(s) that other people should use when attempting to achieve an identical build.

[explanation:factors] There are some conventions about factors that are acceptable to keep constant[hyperlink:L117] -- these include the compiler, compiler version, and the versions of other software that the software depends upon. In contrast, there are other variable factors that we do expect and should accommodate when the software is rebuilt in diverse environments[hyperlink:L97]. It is a good idea to confirm that your software builds reproducibly in those environments too.

Tooling such as reprotest[hyperlink] has been developed to systematically explore the factors that can affect build reproducibility, and we recommend re-using existing utilities rather than writing your own.

Factors that we would like to prevent from affecting the build output

- Software on your computer unrelated to the build process
- date, time
- language and regional settings
- CPU speed, number of cores, load of the build machine
- hostname, user name, build path

Reprotest

[Reprotest] is a tool that rebuilds a project in different environments automatically.

It can apply several variations, including build path, file order, locales, hostname, etc...

It is especially aimed at Debian packages, but it can be configured to also run with other build systems.

Its [readme] offers many examples about how to use it.

[Reprotest](https://salsa.debian.org/reproducible-builds/reprotest)

[readme](https://salsa.debian.org/reproducible-builds/reprotest/-/blob/master/README.rst)

Factors that are usually acceptable to declare as constants

- Toolchain (compiler, ...)
- Dependencies listed in your project

Lockfiles

Some build systems (Go, Cargo, NPM...) allow you to include the exact version of your dependencies.

Whenever possible, you should version it or include it in your source tarballs, so that people will be able to use them to recreate a similar environment to you.

Vendored dependencies

Another possibility is to include a copy of your dependencies in your source tree, or to reference it with similar methods, such as git submodules.

Debian Snapshots

Debian packages must be reproducible with the packages that were availalbe in the archive when they were built.

You can use snapshot.debian.org to create a system in that state.

Document 3:

Title: **Troubleshooting**

A few hints:

* Isolating individual outputs of the build process
* Comparing the content that differs in the build outputs
* If the problem occurred after introducing a variation, then that also provides a hint about potentially-relevant causes

Compare the differing output files, using a tool such as **diffoscope**[link].

We now need to inspect the differences and to try to understand what could have caused each of them.

In some cases, the differences themselves may provide some hints about possible fixes. For example:

| table of examples |

| before | after | seems to be a... | possible remedy

| 2021-04-10 | 2026-09-01 | date | SOURCE_DATE_EPOCH[link] |

https://gitlab.torproject.org/tpo/applications/tor-browser/-/wikis/Reproducible-Builds/Debugging-Android

Document 4:

Title: **Sample use cases**

Part one:

Ask people on the r-b mailing list (and elsewhere) for examples of reproducibility failures and how they were fixed.

All the writeups should have the same outline for consistency. The texts should have some technical details, but should not dive too deeply in the nitty gritty. A potential outline could be something like the following:

How the issue was discovered?

What steps were taken to discover the underlying cause?

How the issue was fixed

The writeups should be fairly brief to make them easy to read. Two or three paragraphs of text for each section is a good amount to aim for.

Part two:

Links to blog posts etc that have similar writeups.

Making the business case for reproducible builds II

- We will send an email to the PQC forum email list

- The from email is not important, one of us will send the email

- We want R-B to be a requirement for FIPS certified implementations

- Maybe we could have help from a cryptographer: perhaps Matthew D. Green

- We should do a bit of research to check other NIST email lists and perhaps more related working groups

- We have to try to convince NIST to test R-B in their certification process

- We should check how NIST runs the validation/certification: e.g., check the [Cryptographic Module Validation Program](https://csrc.nist.gov/projects/cryptographic-module-validation-program)

- All this is a tactic. A single email will not have the work done. So we should do this action as a tactic part of a larger strategy

- We need to figure out who are the "messengers" in the NIST "circle", so that when they read our email and they ask them they are aware of R-B

- For example, if we manage that Bruce Schneier writes a blog post about R-B, this will make it easier to succeed in trying to incorporate this into NIST

- In R-B we are missing a vision. For example, the Tor project has a clear vision: to make people navigate the Internet 100% anonymously. In R-B we have "a clear path from source code to binary", but how does this change the world if achieved? We need this to convince NIST (and other) people.

- One concrete example to find a vision: make the US election machines reproducible

- We could make a list of relevant actors/people (a stakeholders map) that if they understand R-B and support it, they could make a difference

- R-B has an internal vision ("enable anyone to independently verify...") but R-B needs an external vision

- How do we convince investors? They don't care about security. Can we sell R-B without even mentioning R-B? For example, you can avoid a risk of losing all your invest if...

TODO:

- Write down an external vision

- Set clear goals

- Create a stakeholder map

- Create a who/benefits table

Work towards a vision statement:

we want something that can be understood by non-developers, that better captures the values of what r-b provides.

We also got several nice slogans out of this session:

R-B - because life is unpredictable enough

R-B - so content matches the description (on the tin)

R-B - makes assumptions hold true

R-B - to keep devices clean and users healthy

R-B - to ensure that 2+2 is always 4 / never 5 (devrtz likes this one very much)

R-B - so computers do what I want.

R-B - because computers should do, what you tell me/them (to do).

R-B - to extend trust to your digital life

R-B - is FLOSS++

R-B - What you run is what you build (WYRIWYB, lel)

How the food industry talks about "organic"

is how we should talk about RB.

Proposals for a (external) vision statement:

R-B =

Build trustable/trustworthy/transparent/traceable/correct/verifiable/deterministic/hygienic/predictable/guaranteed software

Predictable translation of human intention/instructions (in)to digital action / software / apps / machine instructions.

Extending trust from ... (in)to ...

R-B extends trust from the human readable source code into the machine executable instructions,

so that you can be sure you are running the software you are intending to.

R-B allows everyone to verify machine instructions correspond

to given source code (written {by,for} humans):

Trusting software (authors) is good, being able to verify is better.

Work towards some kind of stakeholder map. The ideas were to:

- Level 0: Create a list of "domains"

- Level 1: For each of these domains, ELI5 the benefits of determinism processes related to that specific domain. To illustrate this, we need to find a way to adapt our talk for each of these domains. The goal of this is to let them want R13Y, or merely, determinism in their processes. This could be separated into several questions like:

  - What is R13Y / Determinism?

  - Why do you want it?

  - How R13Y will help your business?

  - How much is this going to cost?

  - But my stuff is already working, why would I need that?

  - But my stuff is already R13Ble, why would I need that?

- Level 9999: Hide the broccoli in the ice cream

Level 0: Domains:

- Developers
  - They will be harder to attack (Software)
    Question: What is meant here *exactly*?
    Developer machines are more secure if they're running
    tools/development environment that have been verified to be reproducible?
    Something else?!
    - A workflow that is easier to work on individually and collaboratively
  - Enhance cacheability
    - Enhance and reduce the time to reproduce an issue and help to fix quickly the bugs
- Fintech
  - Less chance of money being stolen (FinTech)
- Healthtech
  - More reliable and predictable health appliances (Healthcare)
- Safety/mission-critical software
  - Traceable and debuggable (Safety/mission-critical software)
- Voting systems
  - More transparency, enhance trust (Voting systems)
- Governmental systems (e.g., tax software)
  - More transparency, enhance trust (Governmental systems)
- Scientists / researchers / students
  - Verifiable science, better peer reviews (Scientists, researchers, students)
- A.I.

Assets:

- Powerful Case Studies. (Needed)

Diagram/Illustration of concept (for the homepage)

- Package that looks identical from the outside,
  but when you unpack you get different things:
  sometimes bananas, other times oranges or apples
  => Wouldn't it be nice to know what you're getting?

- Illustrate compilation:
  A chart with 3 components, from left to right:
    - Source code (e.g. represented simply as a file icon)
    - The build process (e.g. illustrated with cog wheel or assembly line)
    - The build artifact (e.g. ".exe")

  The "selling point" than would be:
    With RB you can go from build artifact to corresponding source code
    so you're sure what yo're running

Day 3, Session 2, notes:

convince policymakers instead of businesses

r-b vision / slogan for non developers might need to come from non-developers

fsfe has one slogan "public money public code" which can be understood by anyone, without even knowing what free software is

rename "buy in" to "why does r-b matter"?

|r-b, because you need certainty in the digital world|

benefits for the public / common people:

    certainty

    Investments into source code reviews/audits are not lost

"what you run is what you build"

- R13Y, closing the missing link in supply chain security
- R13Y closing the gap in supply chain security

- R13Y, welding the security chain
- Make the supply chain great again
- Make the supply chain reproducible again
- R13Y, the path to trust
-

Strategic working sessions III

Rebuilders III

(See “Rebuilders II”)

Kernel reproducibility II

No notes were taken for this session

Getting started with reproducible investigations III

See notes for “Getting started with reproducible investigations II”

Making the business case for reproducible builds III

See notes for “Making the business case for reproducible builds II”

Strategic working sessions IV

Rebuilders IV

(See “Rebuilders II”)

Getting Started with Reproducible Builds IV

 * File an issue on the Reproducible Builds Website bugtracker for addition of this "Getting Started" guide.

 * Call for stories in the mailing list: if you documented cases where reproducibility broke for you, can you link from some page? (Yet to be created)

  * Let's include a link to the initial version of the guide on the website

  * [idea] Also include a link to the reproduciblepackage (not 'production code', but provides a lot of examples)

Making the business case for reproducible builds IV

See notes for “Making the business case for reproducible builds II”

Teaching RB at universities

Getting R.B. into universities

ideas for a curriculum (brainstorm result)

Why?

- software integrity
- trust
- the path from code to artifacts
- objectives of RB will help when working together as things as constant environments help trouble shouting
- [to be answered:] "Does this help employability

SWE practices related to R.B.

- understanding of dependency tree
- keep track of your dependencies (versions, lock files, ...)
- Documenting you libraries
- build systems
- CI/CD
- principles for build variance
- the social aspect of building and shipping code

"Theoretic background"

- There is something to do (cc might not be a function in practice)
- readdir order [e.g., in a basic OS lecture]
- "reproducible build commandments"
- different programming languages (compiled, interpreted, ...)
- compression [NB: defined in terms of the decoding function]
- coordination problems?
- cryptography 101 (why SHAs?)

- Information pasing in decentralized neteworks

other

- should add pointer to other sources, in particular to sources describing how to get started on RB

Mapping next conversations and next steps

The group used the final session to identify areas for ongoing collaboration. A focal point in this discussion was a visioning exercise enumerating 3 lists we will review at the next Reproducible Builds Summit.

The prompt to the group was posed as “When we get to the 2025 Reproducible Builds Summit what MUST/SHOULD/COULD be true?”

The idea was to enumerate baseline goals for the next year (“MUST”), additional high-priority goals (“SHOULD”), and stretch goals (“COULD”).

The following must be true by R-B summit 2025:

- common attestation format for reproducibility events
- at least 1 language package manager is integrated into rebuilderd
- 3 solid case-studies of r13y solving a problem in the wild
- Arch Linux is Hunnid percent 100% reproducible
- Debian rebuild system fully running
- Make the r-b.org website accesible to non-tech people
- Debian Rebuilder Network
- Must ?icon? Badgers
- jvm reproducibility
- available data for most of Maven central
- Integrated with Rep Central
- We need more concrete use-cases
- R-B.o must have a list of ways to solve frequent problems
- formal definitions
- Nix

The following should be true by R-B summit 2025:

- summit should be a more diverse group
- openSUSE is over 99% reproducible
- All distros should have build recipes (aka buildinfo) and build deps (aka snapshot) available for rebuilds

- diffoscope has a beautiful API
- OSS rebuild, repro central and rebuilderd produce compatible evidence of rebuilding
- TorBrowser releases include some build attestation
-
- Concrete procurement guidelines for at least one major company or gov require R-B
- 100% reproducible minimal Arch Linux install possible
- support more maven repositories such as jet-brains-dev
- Every rebuilder should publish build-attestations for their builds
- Top 100 highest popcon packages reproducible in Debian
- a new R-B T-shirt
- There is a good "getting started" tutorial with many real-world diffs and examples (e.g. https://github.com/bmwiedemann/theunreproduciblepackage/)
- Debian: <300 pkgs unreproducible
- Linux kernel builds reproducibly in most major distros
- promote r13y to students
- Three (or more) Debian rebuilders running continously

The following could be true by R-B summit 2025:

- "Sell" R-B to the Quantum Computing community
- Establish a "commonly recognized" badge that shows a project is R-B
- More build-tools maintainers could join the next summit
- repro-env preinstalled on github actions image
- R-B becomes requirement in a new standard
- PyPi packages have been assessed for reproducibility
- have a full course content
- The average programmer is aware that reproducible builds is a good thing
- TorBrowser updater is verifying build-attestations before applying updates

Appendix I – Additional notes from “RB Problem Prioritization Session”

-- Bernhards list:

The further down an entry, the closer it is to completion (become reproducible in Factory)

ToDo

send https://invent.kde.org/qt/qt/qtbase/-/merge_requests for https://codereview.qt-project.org/c/qt/qtbase/+/494174 backport

zls version_data in debug

mayavi zip mtimes, core-count,other .py diffs

python-pydata-sphinx-theme

git-credential-keepassxc rust codegen-units, low-entropy

git-annex parallelism

rmt-server https://bugzilla.opensuse.org/show_bug.cgi?id=1227542 random timing bug? dir permission issue - not reproducible anymore after 2d? - use dm-delay to reproduce?

zola unknown rust/llvm

turbo minor ASLR

python-contourpy random from mesonpy?

Fragments random rust/llvm

python-pyrage random rust/llvm .so +other?

openttd minor,ASLR

mingw64-runtime minor .a

ecl random (LISP) via ecl-24.5.10/build/clos/boot.c

shadowsocks-rust https://github.com/AlephAlpha/build-time/pull/5 date+time ; rewritten to use bad build_time crate

binutils low-entropy PGO ; -readdir? asm see ~/osc/t/openSUSE:Factory:Rings:0-Bootstrap # /usr/lib/build/pkg-diff.sh -a _build.standard.x86_64/binutils/binutils-2.42-0.x86_64.rpm _build.1/binutils/binutils-2.42-0.x86_64.rpm

#wireplumber parallelism, low-entropy?

python-libarchive python-praatio python-pytest-attrib nochecks https://bugzilla.opensuse.org/show_bug.cgi?id=1222349 python-textX FTBFS-nochecks

qqc2-breeze-style6 parallelism

kf6-qqc2-desktop-style qt6-declarative toolchain issue? parallelism?

cmake CMake.qch

stl-thumb rust/llvm

python-orjson rust/gcc13 asm

eww rust/llvm

openssl-1_0_0 https://bugzilla.suse.com/show_bug.cgi?id=1219879 obj_dat.h

python-pyarrow ASLR

app-builder minor build-id

cockpit filesys (python) -> fixed in 324 upstream

javadoc toolchain filesys-readdir-order: antlr4 apache-commons-csv ...

-> https://tests.reproducible-builds.org/debian/issues/unstable/random_order_in_documentation_generated_by_javadoc_issue.html

openjfx filesys + minor mtime/other?

cuarzo-srm filesys

python-seaborn filesys low-entropy .pyc python3.9

python-debugpy filesys low-entropy .pyc python3.9

parboiled minor jar mtimes

scala filesys

#deepin-gir-generator

scummvm minor

python-frozenlist https://github.com/cython/cython/issues/5949 random path, toolchain random tmp path from https://github.com/sysfce2/python-frozenlist/blob/master/packaging/pep517_backend/_backend.py#L199

plasma5-workspace parallelism

gri ps

deluge .egg zip mtime

julia CPU in openblas + ordering + other = parallelism?

dbus-sharp2 mono (ignored by build-compare+monodis)

calibre CPU-dependent from Qt6 QImage scaler | .png files differ from SSE-4.1

ceph boost b2 project-cache.jam filesys-readdir-order

contrast unknown rust/llvm

dealii doxygen parallelism?

deno rust zstd-sys?

difftastic parallelism order? + rust/llvm

#dmd parallelism + ASLR

eclipse .jar files containing .jar files vary in size

elixir-ex_doc

gammaray .qch order

ghc sphinx doctrees + ghc-9.6.3/GHC/Runtime/Interpreter.dyn_hi

groovy18 order issue in .class file

installation-images various issues from efi + %post

java-21-openjdk parallelism+ASLR FTBFS-2034 =
https://bugzilla.opensuse.org/show_bug.cgi?id=1213796, date

java-21-openj9 date TR_BUILD_NAME + other

jedit javadoc .html date ; build.xml needs modificationtime="2000-01-01T00:00:00Z"

kdevelop5-plugin-php parallelism+ASLR in libkdevphpparser.so via phpparser.cpp

ksh year in copyright ; compile time benchmarking ; SIGQUEUE_MAX detection

lapce filesys-readdir-order minor

lazarus

ldc order x+?ASLR

libaom .png, copyright year in aom_version.h

libqt5-qtdoc .qhp, .qch (sqlite3), doc order

nodejs20 v8_snapshot_blob_data varies

nodejs21

pass-import ASLR order issue - toolchain pandoc?

pympress date+time from python-Babel?

python-cPyparsing cpython toolchain fix in :test?

python-HyperKitty variations in
/srv/www/webapps/mailman/hyperkitty/static/CACHE/css/output.*.css from Django?

python-mesonpep517 cpython

python-pandas OBS verification fail - ?from preinstallimage?

python-scikit-learn parallelism via cython+openmp - similar to https://github.com/yt-project/yt/issues/4611

python-scipy ....

qgis parallelism (via ninja?)

#qt-creator parallelism in libTracing.so.11.0.2? gone?

rage-encryption rust/llvm?

sad order ?from rust/llvm?

scap-security-guide order, https://bugzilla.opensuse.org/show_bug.cgi?id=1230361 date

sdcc 32bit epoch timestamp

smlnj order + EPOCH timestamp

stl-thumb unknown rust/llvm

swipl prolog date + FTBFS-2029 + other

tiny rust/llvm

virtualbox copyright year ; _BUILD_STRING date+time ; tar mtime ; g_VTGObjHeader https://www.virtualbox.org/ticket/16854 + https://sourceforge.net/p/gsoap2/patches/185/

warp filesys maybe https://github.com/briansmith/ring/issues/1625

wpewebkit copyright year ; varying JSCBytecodeCacheVersion

xemacs order in lisp/custom-load.el ; .elc ; other

xemacs-packages readdir+ASLR+CPU in .elc order ; SR 1119571 partial date

Analyzed:

librsvg FTBFS-cpu

asymptote toolchain pdflatex date

ginac pdflatex

glucat pdflatex

#unison pdf from pdfTeX - cannot reproduce after 20231122?

openmpi5 https://github.com/open-mpi/ompi/pull/11847 date+readdir

coq parallelism from ocaml +maybe others - same as ocaml-camlp-streams?

grass https://github.com/OSGeo/grass/pull/3415 https://github.com/OSGeo/grass/pull/3417 https://github.com/OSGeo/grass/issues/3038 date+parallelism in r3.mapcalc

hevea random ocaml tmp dir from ocamlopt mathgl random example .png

R-base embedded random tmp path in /usr/lib64/R/library/KernSmooth/help/paths.rds; many other

rabbitmq-server timestamps in zip from elixir toolchain

rakudo patch-mtime + https://github.com/rakudo/rakudo/issues/5427 + #https://github.com/rakudo/rakudo/pull/5426

wine parallelism, (?needs mingw-binutils update according to msmeissn) ; timestamp in PE .dll + #https://gitlab.winehq.org/wine/wine/-/merge_requests/4035

python-pytest-httpbin FTBFS SSL + other

?MozillaFirefox FTBFS-j1

clisp hostname + uninitialized mem dump

scheme48 unreproducible scheme dump

chezscheme scheme .boot files

emacs .eln ASLR + .pdmp unknown issue (5+6 random bytes)
https://lists.gnu.org/archive/html/emacs-devel/2024-01/msg00464.html
https://mail.gnu.org/archive/html/emacs-devel/2024-02/msg00417.html
https://mail.gnu.org/archive/html/emacs-devel/2024-05/msg01026.html

maxima clisp .mem dump

pgloader from sbcl (lisp dump)

scsh totally unreproducible scheme image dump

spack parallelism https://github.com/spack/spack/pull/36064

racket various issues - unknown source - scheme dump

xindy xindy.mem CLISP memory image data dump

python-numba toolchain:

- python-iminuit numba toolchain
- python-librosa numba toolchain
- python-xarray-einstats numba toolchain

https://trello.com/c/lXGJn6Wf/44-ghostscript

gfan

LiE

lout

pnetcdf

srecord

reported in openSUSE:

dlib https://bugzilla.opensuse.org/show_bug.cgi?id=1223168 CPU

perl https://bugzilla.opensuse.org/show_bug.cgi?id=1230137 records kernel version

systemd https://bugzilla.opensuse.org/show_bug.cgi?id=1228091 pesign verification issue

kf6-kirigami https://bugzilla.opensuse.org/show_bug.cgi?id=1228131 race parallelism

python-torch https://bugzilla.opensuse.org/show_bug.cgi?id=1229033 CPU

hawk2 https://bugzilla.opensuse.org/show_bug.cgi?id=1230275 date, sprockets .cache files vary for unknown reason

crash https://bugzilla.opensuse.org/show_bug.cgi?id=1230281 parallelism, race

nauty https://bugzilla.opensuse.org/show_bug.cgi?id=1225415 CPU

python-mpi4py https://bugzilla.suse.com/show_bug.cgi?id=1212698 from gcc?

tycho + eclipse-jgit https://github.com/openSUSE/build-compare/issues/65 bogus

mozilla-nss https://bugzilla.opensuse.org/show_bug.cgi?id=1081723 DSA sig

wsl-appx https://bugzilla.opensuse.org/show_bug.cgi?id=1189698 verification failure - publisher from OBS ssl cert

libguestfs https://bugzilla.opensuse.org/show_bug.cgi?id=1216986 embeds host /etc/hosts in supermin.d/base.tar.gz ; created.rid date (minor/strip-nd)

python-numpy https://bugzilla.opensuse.org/show_bug.cgi?id=1216458 random file names

libkolabxml https://bugzilla.opensuse.org/show_bug.cgi?id=1060506 ASLR/CPU order-issue in xsdbin ; partially workarounded?

grub2 https://bugzilla.opensuse.org/show_bug.cgi?id=1217967 https://bugzilla.opensuse.org/show_bug.cgi?id=1217619 grub.xen tar mtime + (parallelism?) + filesys-readdir-order(from grub_util_fd_readdir) + octal UNIX timestamp from grub-mkstandalone | grub-mkimage

ovmf-riscv64-code.bin

#qt6-webengine https://bugzilla.opensuse.org/show_bug.cgi?id=1217774 qtpdf.index parallelism

libcamera https://bugzilla.opensuse.org/show_bug.cgi?id=1217690 random: private key signatures of modules elixir https://bugzilla.opensuse.org/show_bug.cgi?id=1205134 FTBFS-j1 stuck

llvm13 https://bugzilla.opensuse.org/show_bug.cgi?id=1195427 ASLR

kubefirst https://bugzilla.opensuse.org/show_bug.cgi?id=1221680 bug

openssl-3 https://bugzilla.opensuse.org/show_bug.cgi?id=1223336 random debugsource

kernel-default https://bugzilla.opensuse.org/show_bug.cgi?id=1230414 BTF variations https://www.kernel.org/doc/html/latest/bpf/btf.html https://rb.zq1.de/compare.factory-20240904/diffs/kernel-default-compare.out

python310 https://bugzilla.opensuse.org/show_bug.cgi?id=1040589 - improvable

python311

python312

python38

python39

gcc12

bash

reported upstream:

seahorse https://gitlab.gnome.org/GNOME/seahorse/-/issues/394 parallelism, bug (@VCS_TAG@)

python-ruff https://github.com/astral-sh/ruff/issues/12169 python-ruff ASLR

warzone2100 https://github.com/BinomialLLC/basis_universal/issues/374 parallelism

starship https://github.com/starship/starship/pull/5352 1-byte-diff in .note.ABI-tag ?from rust/llvm?

gcc14 https://gcc.gnu.org/bugzilla/show_bug.cgi?id=114895 https://bugzilla.suse.com/show_bug.cgi?id=1223169 FTBFS-2038

llvm11-17 https://bugzilla.opensuse.org/show_bug.cgi?id=1199076

llvm17/rust1.73 https://github.com/rust-lang/rust/issues/128675 https://github.com/llvm/llvm-project/issues/72206

gstreamer-plugins-rs https://gitlab.freedesktop.org/gstreamer/gst-plugins-rs/-/issues/599 date ; unknown issue in pkgconfig .pc generation

efl https://git.enlightenment.org/enlightenment/efl/issues/41 parallelism

enlightenment-theme-dark

enlightenment-theme-openSUSE

enlightenment-theme-openSUSE-ice

enlightenment-theme-openSUSE-neon

enlightenment-theme-openSUSE-oliveleaf

libpinyin https://github.com/libpinyin/libpinyin/issues/162 ASLR

fcitx5-zhuyin

fcitx-libpinyin

ibus-libzhuyin

https://github.com/mono/mono/issues/20172 report unknown/time-based nondeterminism

keepass

mono-addins

mono-core

nunit

flickrnet mono PE-header-timestamp, 1024-bit ?PublicKeyToken?, other ; ignored by build-compare+monodis

kernel-source:kernel-docs parallelism/race bug https://lore.kernel.org/linux-doc/33018311-0bdf-4258-b0c0-428a548c710d@suse.de/T/#t https://github.com/sphinx-doc/sphinx/issues/2946 = https://github.com/sphinx-doc/sphinx/issues/6714 ; workaroundable

(also guake, qemu listed elsewhere (already fixed in Factory))

ocaml-camlp-streams https://github.com/ocaml/dune/issues/9152 https://github.com/ocaml/camlp-streams/issues/9

ocaml-extlib

release-notes-openSUSE toolchain https://github.com/apache/xmlgraphics-fop/pull/65 date+random-ID gsequencer "

bibletime "

pam:full .pdf from fop

erlang " and other issues

sbcl https://sourceforge.net/p/sbcl/mailman/sbcl-devel/thread/3ebdd95c-c498-462f-9cfe-7d05a1ee0044%40suse.de/ report timestamp+other

lilypond https://sourceforge.net/p/testlilyissues/issues/5328/ https://savannah.gnu.org/patch/?9370 https://codereview.appspot.com/337650043

python-yt https://github.com/yt-project/yt/issues/4611 parallelism +#=> https://github.com/yt-project/yt/pull/4609 python filesys order

wrk https://github.com/wg/wrk/issues/507 hash order issue

python3-pyside6 ASLR order from shiboken6

python3-pyside2 https://bugreports.qt.io/browse/PYSIDE-2508 order

rash https://github.com/willghatch/racket-rash/issues/52 report parallelism-related non-determinism

gnu-cobol https://savannah.gnu.org/bugs/index.php?54361 https://bugzilla.opensuse.org/show_bug.cgi?id=1195087

bcc order in bcc-lua from toolchain luajit https://github.com/LuaJIT/LuaJIT/issues/1008

boost https://github.com/boostorg/boost/issues/741 2038-date influences order in libboost_log.so.1.82.0

joker https://github.com/candid82/joker/issues/491 hash random order # https://github.com/candid82/joker/pull/490 sort, partial

colord https://github.com/omgovich/colord/issues/122 https://bugzilla.opensuse.org/show_bug.cgi?id=1217747 .icc only-CPU

neovim https://github.com/neovim/neovim/issues/26387 order issue maybe from gen_api_dispatch.lua

cosmic-edit https://github.com/pop-os/cosmic-edit/issues/221 hash order issue, rust

soapy-sdr https://github.com/pothosware/SoapySDR/issues/428 parallelism

qt6-virtualkeyboard qt6-declarative https://bugreports.qt.io/browse/QTBUG-121643 bug? parallelism

qt6-quick3d parallelism https://bugreports.qt.io/browse/QTBUG-122722 https://bugreports.qt.io/browse/QTBUG-121643 xiphos/yelp-tools https://gitlab.gnome.org/GNOME/yelp-tools/-/issues/24 toolchain .epub content UUID

pspp https://savannah.gnu.org/bugs/index.php?65485 bug,parallelism /usr/share/info/pspp.info-1.gz # unreproducible testsuite.log => dropped

fix submitted upstream:

TeXmacs https://github.com/texmacs/texmacs/pull/77 date

1188555 1188166 cloudflared https://github.com/cloudflare/cloudflared/pull/1289 date

mage https://github.com/magefile/mage/pull/474

--rpm + obs-build https://bugzilla.opensuse.org/show_bug.cgi?id=1206816 https://github.com/rpm-software-management/rpm/pull/2762 https://github.com/rpm-software-management/rpm/issues/2343 + https://github.com/openSUSE/obs-build/blob/1fe25db/build-recipe#L259 toolchain ; TODO

kernel-obs-build patch per mail - initrd cpio varies in /etc/hosts from build machine hostname

python-PyGithub https://github.com/PyGithub/PyGithub/pull/3045 FTBFS 2024-11-25

fix merged upstream:

gnutls https://gitlab.com/gnutls/cligen/-/merge_requests/5 date >3.8.6

less https://github.com/gwsw/less/pull/567 date >661

ooRexx https://sourceforge.net/p/oorexx/bugs/1712/ ASLR ; fixed in 5.0.0?

OpenRGB https://gitlab.com/CalcProgrammer1/OpenRGB/-/merge_requests/2103 filesys >0.9

qpid-proton https://github.com/apache/qpid-proton/pull/411 sort >0.39

goldendict-ng https://bugreports.qt.io/browse/QTBUG-115737 qmake toolchain ; also rewritten in goldendict-ng >23.05.03

whatsie https://github.com/keshavbhatt/whatsie/pull/146 date (needs qmake patch https://codereview.qt-project.org/c/qt/qtbase/+/494174 in qt5)

qutebrowser https://github.com/qutebrowser/qutebrowser/pull/8233 FTBFS-2036

mhvtl https://github.com/markh794/mhvtl/pull/128 tar >1.71

python311:doc https://bugzilla.opensuse.org/show_bug.cgi?id=1227999 date #https://github.com/python/cpython/pull/121872 python311:doc date, audit_events.html https://github.com/python/cpython/issues/121874 html race bug, gzip header mtime https://github.com/sphinx-doc/sphinxcontrib-devhelp/pull/13 https://github.com/sphinx-doc/sphinx/pull/12606 nautilus https://gitlab.gnome.org/GNOME/nautilus/-/merge_requests/1555 date from data/meson.build date -I call >46.2

rmw https://github.com/theimpossibleastronaut/rmw/pull/444 https://github.com/theimpossibleastronaut/rmw/issues/439 FTBFS-2038 >0.9.2

SR:

-1119524 occt sort (not upstream)

1192491 rpm-config-SUSE date

-1194375 agama-integration-tests https://github.com/openSUSE/agama/pull/1576 random port in .lock file

1199220 runawk nochecks

Done:

#1130635 1130399 python-efl Spinx doctrees pickle

#1119096 kanku https://github.com/maros/MooseX-App/pull/71 toolchain, date from perl-MooseX-App => SR 1119096

#1108744 siproxd https://build.opensuse.org/request/show/1108744

#1117880 deepin-desktop-schemas

#1117898 python-quantities date

#1123416 openblas cpu count (orig SR 1118201)

#1124071 1123507 1118130 spack https://github.com/spack/spack/pull/36064 parallelism ; solved differently upstream

#1121668 ghc* https://github.com/opensuse-haskell/ghc-rpm-macros/pull/1 toolchain >2.5.3

#1124263 kitty https://github.com/kovidgoyal/kitty/pull/6685 sort+mtime >0.30.1

#1123597 spotifyd https://github.com/Spotifyd/spotifyd/issues/1167 fixed in 0.3.5

#1120220 selinux-policy/policycoreutils https://bugzilla.opensuse.org/show_bug.cgi?id=1216452 toolchain, order issues in /usr/share/selinux/devel/html/index.html

#1127255 google-noto-fonts https://github.com/notofonts/Arimo/pull/17 downstream patch fix caused mtime to creep into build results

#1127661 xen date+time

#1125191 amber-cli date

#1127547 guake parallelism/race

#1129136 ipxe https://github.com/ipxe/ipxe/pull/1082 .sdsk random from mtools init_random(time)

#1129609 pcp varying HAVE_MEMORY_H

? groff time in .pdf

#apache-arrow https://github.com/apache/arrow/issues/37276 FTBFS-j1

#1129751 fdo-client https://bugzilla.opensuse.org/show_bug.cgi?id=1216293 private key

#1127368 guile-fibers parallelism https://issues.guix.gnu.org/issue/20272 https://bugzilla.opensuse.org/show_bug.cgi?id=1170378

#1127367 guile-newt parallelism

#1130552 procmail benchmarking

#1130719 1129495 1129303 quazip missing fonts quazip .png - toolchain random from doxygen->dot

#1129040 tigervnc drop RSA-signature https://bugzilla.opensuse.org/show_bug.cgi?id=1208478 tigervnc RSA key

#1130500 1128769 gutenprint date+hostname https://sourceforge.net/p/gimp-print/source/merge-requests/9/ date >5.3.4

#1132004 1131679 python-pyface time = https://github.com/enthought/pyface/issues/1254

#1135210 1134362 xemacs toolchain gnugo+uim hostname

#1132811 python-rdflib https://github.com/RDFLib/rdflib/issues/2645 random Graph identifier ? toolchain from python-Sphinx?

#1132690 1132599 python-yarl https://github.com/cython/cython/issues/5949 cpython random path - regression in 1.9.3

#1136570 orthanc-ohif https://orthanc.uclouvain.be/hg/orthanc-ohif/rev/154cb76a042f 1.1 after 2023-08

#1137474 python-rjsmin drop gcc instrumentation

#1140578 doxygen toolchain libzypp+wxWidgets-3_2:doc filesys clhep

#1138193 1138082 libjcat https://bugzilla.opensuse.org/show_bug.cgi?id=1218715 random/crypto

#1141433 mumble https://github.com/mumble-voip/mumble/pull/6147 filesys >1.5.517

#1162930 1137333 warewulf https://bugzilla.opensuse.org/show_bug.cgi?id=1217973 cpio mtime + inode

#1137377 hub https://github.com/mislav/hub/pull/3344 random

#1144993 grub2 filesys in grub.efi

#1147594 ocaml-dune https://github.com/ocaml/dune/issues/9794 filesys ; merged: https://github.com/ocaml/dune/pull/9735 https://github.com/ocaml/dune/issues/9507 parallelism >= 3.14

#1151123 1150775 latex2html drop latex log

#1139319 lagrange https://codeberg.org/skyjake/lagrange/pulls/3 zip mtime 1.17.6

#osc https://github.com/openSUSE/osc/pull/1455 https://github.com/openSUSE/osc/pull/1454 build-jobs captured in man-page

#intel-graphics-compiler https://github.com/intel/intel-graphics-compiler/issues/302 https://github.com/intel/intel-graphics-compiler/commit/4354d0bb3b8d1cd436b6601327c076abfcf9d2ff 20231025+

#mame order issue from https://github.com/mamedev/mame/pull/11651 + mame.make filesys-readdir-order from genie + other ; |(premake4 had order issues?) ; => SR 1119553

#1146850 rehex https://github.com/solemnwarning/rehex/pull/221 zip mtime >0.60.1

#1150565 maildir-utils https://github.com/djcb/mu/issues/2570 >=1.11

kernel-vanilla .vmlinuz.hmac ; Build time autogenerated kernel key0 = https://bugzilla.suse.com/show_bug.cgi?id=1187167

kernel-default " - becomes reproducible with _projectcert.crt in checkout

1155067 rabbitmq-java-client maven timestamp Bnd-LastModified

1160385 orthanc-volview sort+mtime https://orthanc.uclouvain.be/hg/orthanc-ohif/rev/154cb76a042f after 2023-08 >1.1

1166080 1163266 musique date (year) from qmake _DATE_

1164244 Setzer https://github.com/mesonbuild/meson/pull/12790 https://github.com/mesonbuild/meson/pull/12788 filesys / meson toolchain >=1.4.0 #meson https://github.com/mesonbuild/meson/pull/12528 toolchain, sort >=1.3.1

1154566 presenterm https://github.com/mfontanini/presenterm/pull/202 filesys (rust) >0.6.1

buildah https://github.com/containers/buildah/issues/5191 timestamp 4 byte EPOCH vary in github.com/containers/buildah/internal/mkcw..gobytes.1

#gap https://github.com/gap-system/gap/pull/5551 https://github.com/gap-system/gap/pull/5550 date fixed in 4.12.2

scipy/pythran https://github.com/serge-sans-paille/pythran/pull/2131 toolchain, date/time

1168496 go1.13 parallelism

1167936 go1.14 parallelism

1168495 go1.15 parallelism

1168439 1168020 ollama https://github.com/ollama/ollama/pull/2836 gzip mtime >0.1.27

1169140 1168486 postfish benchmarking / toolchain fftw3 https://github.com/FFTW/fftw3/issues/337

1163788 1163778 nfdump https://github.com/phaag/nfdump/pull/482 date - fix in 1.7.4

.1139511 1133453 1121011 qemu parallelism-issue from sphinx ; + https://gitlab.com/qemu-project/SLOF/-/merge_requests/1 date + #https://github.com/openSUSE/qemu/pull/44 hostname ; OBS/osc-FTBFS https://bugzilla.opensuse.org/show_bug.cgi?id=1221340

.1169622 1166592 pommed https://salsa.debian.org/mactel-team/pommed/-/merge_requests/2 parallelism

1169977 geany/glfw https://github.com/geany/geany/pull/3785 https://github.com/geany/geany/issues/3717 toolchain geany glfw.c.tags >geany-2.0

ovmf https://github.com/tianocore/edk2/pull/5845 https://github.com/tianocore/edk2/pull/5550 https://bugzilla.opensuse.org/show_bug.cgi?id=1217704

1173445 kraft https://github.com/dragotin/kraft/pull/215 hostname > 1.1

git-interactive-rebase-tool https://github.com/MitMaro/git-interactive-rebase-tool/pull/881 >2.3.0 # not in Factory

1188352 1188059 1187694 gettext-runtime jar mtime

1188675 1188550 gegl https://gitlab.gnome.org/GNOME/gegl/-/issues/337 parallelism, memory

1189530 1188512 latex2html nochecks

1188305 1188204 librcc https://github.com/RusXMMS/librcc/pull/5 date >0.2.13

1175022 gitui https://github.com/extrawurst/gitui/pull/2202 date >= 0.26.2

1178603 python-oslo.messaging https://github.com/openstack/oslo.messaging/commit/dc55d64df989bdb5161ca8ad8d74115cc2959174 hostname in doc >= 14.8.0

python-pydantic-core https://github.com/pydantic/pydantic-core/pull/1221 FTBFS-2032 >2.16.3

kubernetes1.24 https://github.com/golang/go/issues/63851 = https://github.com/kubernetes/kubernetes/issues/110928 kubernetes1.25

1190458 1190449 kubernetes1.26

1190320 openblas https://bugzilla.opensuse.org/show_bug.cgi?id=1228177 cpu type in openblas.pc

1190247 1187675 libdb-4_8 jar mtime, needs upstreaming

1190940 1190278 xorg-x11-fonts:converted #https://gitlab.freedesktop.org/xorg/app/fonttosfnt/-/merge_requests/22 toolchain from fonttosfnt - timestamp in macTime

1190272 1188202 armagetron https://gitlab.com/armagetronad/armagetronad/-/merge_requests/162 date >0.2.9.2.3 https://gitlab.com/armagetronad/armagetronad/-/tags

1191150 1189287 libreoffice https://lists.reproducible-builds.org/pipermail/rb-general/2023-November/003121.html various: clucene .idxl ; order ; copyright year ; records -jN, jar mtime ; => SR 1188998 1188447 clucene-core

1191822 1191536 python-EditorConfig .pyc mtime from %check

1192457 1190933 pop-launcher https://github.com/pop-os/launcher/issues/230 parallelism https://github.com/openSUSE/post-build-checks/pull/65 toolchain, avoid rust parallelism

1194839 lapackpp https://github.com/icl-utk-edu/lapackpp/pull/68 hostname

1194838 blaspp https://github.com/icl-utk-edu/blaspp/pull/87 hostname

#python-grpcio https://github.com/grpc/grpc/pull/35687 sort filesys

1195841 lua-lmod https://github.com/TACC/Lmod/pull/702 date

systemd https://github.com/systemd/systemd/pull/31080 sort filesys

1198030 1192626 ca-certificates-mozilla:ca-certificates-mozilla-prebuilt https://bugzilla.opensuse.org/show_bug.cgi?id=1229003 date in java-cacerts

#waf minor: time-based .pyc

obsolete:

java-11-openj9

java-1_8_0-openjdk

#kopete https://invent.kde.org/network/kopete/-/merge_requests/14

WONTFIX:

update-test-trivial