

Software Heritage

Analyzing the Global Graph of Public Software Development

Stefano Zacchiroli

Université de Paris & Inria — zack@upsilon.cc, [@zacchiro](https://twitter.com/zacchiro)

19 May 2021

Team ACES — Télécom Paris
(online)



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

About me

- Associate Professor (*Maître de conférences*), Université de Paris
- on leave (*délégation*) at Inria
- Free/Open Source Software activist (20+ years)
- Debian Developer & Former 3x Debian Project Leader
- Former Open Source Initiative (OSI) director
- Software Heritage co-founder & CTO



- Associate Professor (*Maître de conférences*), Université de Paris
- on leave (*délégation*) at Inria
- Free/Open Source Software activist (20+ years)
- Debian Developer & Former 3x Debian Project Leader
- Former Open Source Initiative (OSI) director
- Software Heritage co-founder & CTO

Research path

- 1 Formal methods for ensuring the quality of software upgrades (Mancoosi project)
Industry adoption: Debian, OPAM, Eclipse P2
- 2 Formal methods for automated upgrade planning in the cloud (Aeolus project)
Industry adoption: Mandriva, Kyriba
- 3 Large-scale software evolution analysis (Debsources platform)
- 4 Very-large-scale source code analysis and preservation (Software Heritage)
→ this talk

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Security synergies and outlook



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog



find and reference all
software source code



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog



find and **reference** all
software source code

Universal archive



preserve all software
source code



Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Collect, preserve and share *all* software source code

Preserving our heritage, enabling better software and better science for all

Reference catalog



find and **reference** all
software source code

Universal archive



preserve all software
source code

Research infrastructure



enable analysis of all
software source code

An international, non profit initiative

Sharing the vision



United Nations
Educational, Scientific and
Cultural Organization



www.softwareheritage.org/support/testimonials

Donors, members, sponsors



Platinum sponsors



Gold sponsors



Silver sponsors



Bronze sponsors



Archiving goals

Targets: VCS repositories & source code releases (e.g., tarballs, packages)

We DO archive

- file **content** (= blobs)
- **revisions** (= commits), with full metadata
- **releases** (= tags), ditto
- where (**origin**) & when (**visit**) we found any of the above

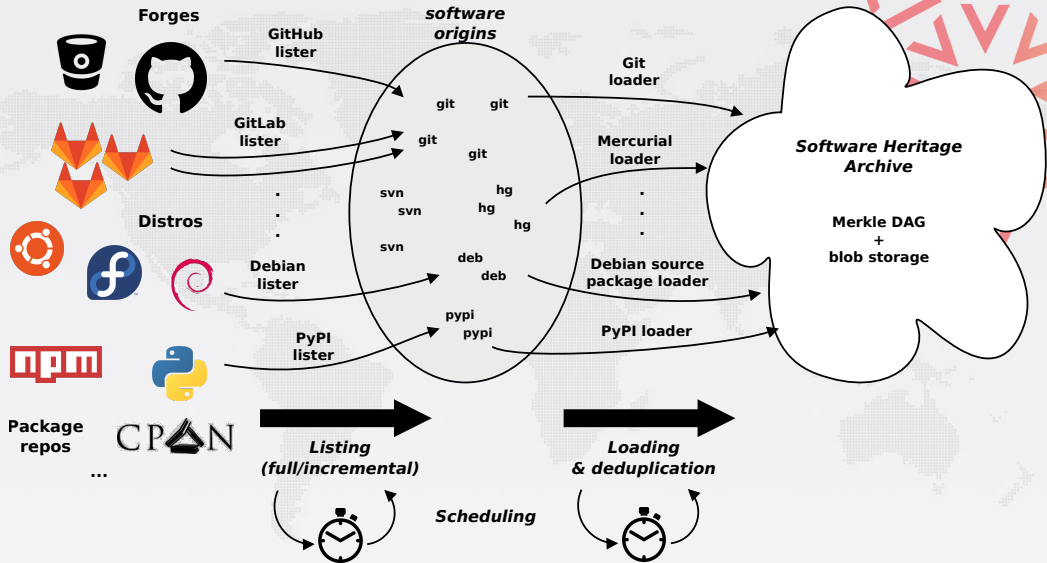
... in a VCS-/archive-agnostic **canonical data model**

We DON'T archive (yet)

- homepages, wikis
- BTS/issues/code reviews/etc.
- mailing lists

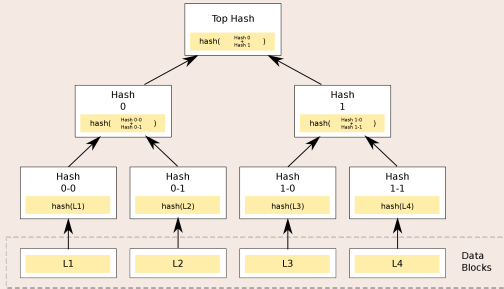
Long term vision: play our part in a *"semantic wikipedia of software"*

Data flow



Merkle trees

Merkle tree (R. C. Merkle, CRYPTO 1987)

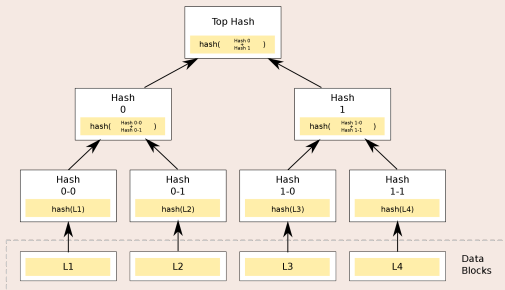


Combination of

- tree
- hash function

Merkle trees

Merkle tree (R. C. Merkle, CRYPTO 1987)

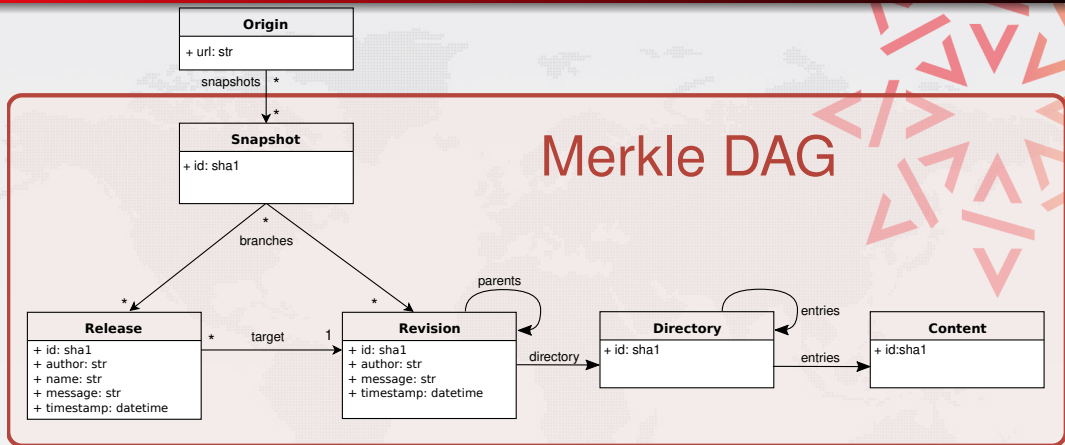


Combination of

- tree
- hash function

Classical cryptographic construction

- fast, parallel signature of large data structures
- widely used (e.g., Git, blockchains, IPFS, ...)
- built-in deduplication



A **global graph** linking together fully **deduplicated** source code artifact (files, commits, directories, releases, etc.) to the places that distribute them (e.g., Git repositories), providing a **unified view** on the entire *Software Commons*.

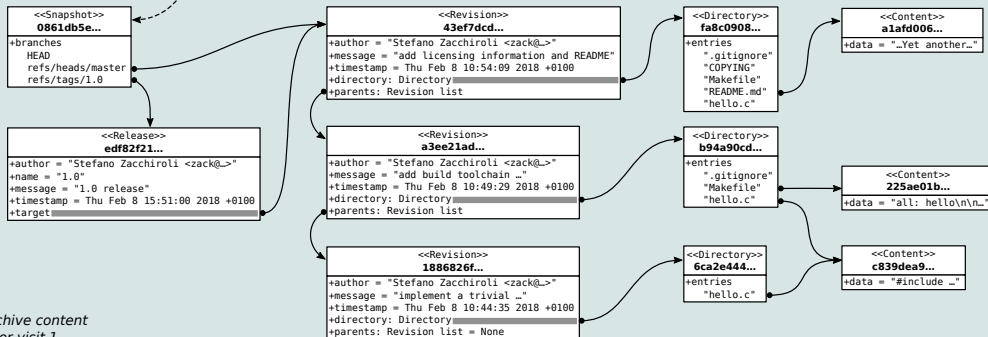
The archive: a (giant) Merkle DAG

origin
https://forge.softwareheritage.org/source/helloworld.git

visit
1

snapshot
0861db5e...

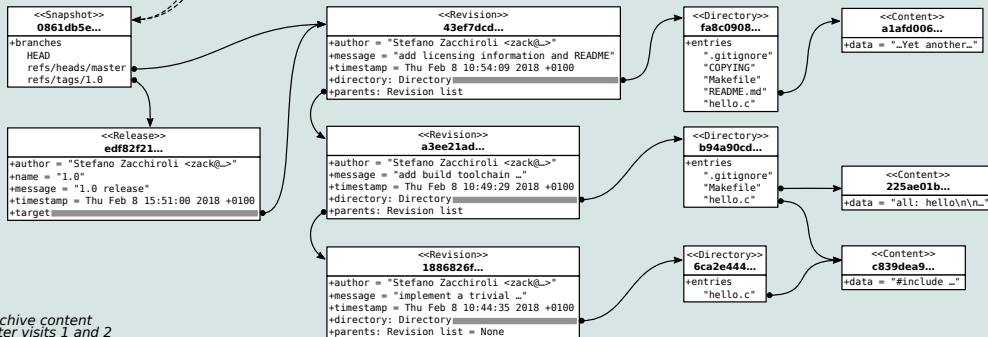
timestamp
Fri Feb 9 12:38:45 2018 +0100



Archive content
after visit 1

The archive: a (giant) Merkle DAG

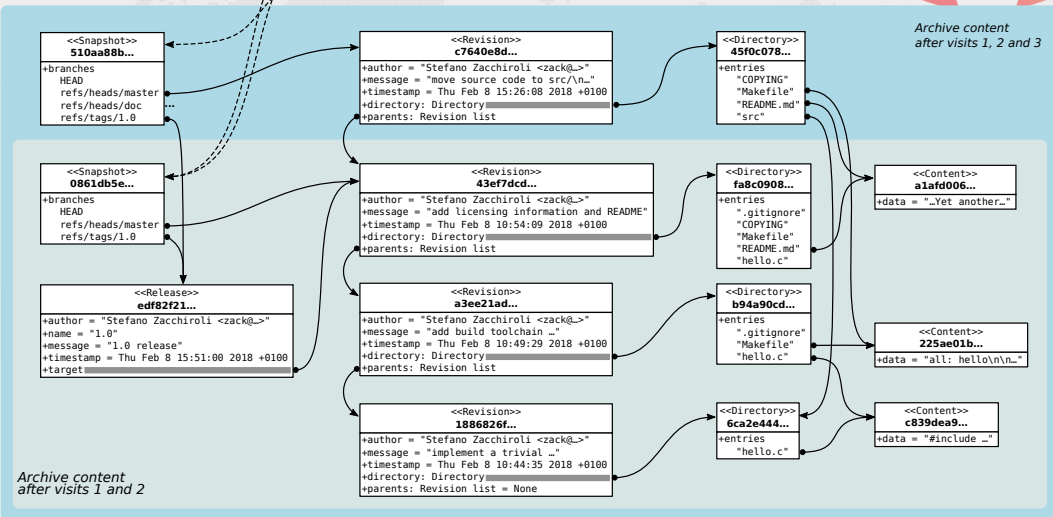
origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	2	0861db5e...	Fri Feb 9 13:29:00 2018 +0100



Archive content
after visits 1 and 2

The archive: a (giant) Merkle DAG

origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	2	0861db5e...	Fri Feb 9 13:29:00 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	3	510aa88b...	Fri Feb 9 15:52:50 2018 +0100





Bitbucket



debian

Framagit

GitHub



GitLab

GITORIOUS

Google code



GNU

Guix

HAL
archives-ouvertes.fr

Inria
Institute of Mathematics and Computer Science

IPOL Journal

npm

NixOS

python
python.org



Bitbucket



debian

Framagit

GitHub



GitLab

GITORIOUS

Google code



GNU

Guix

HAL

archives-ouvertes.fr

Inria

Recherche en informatique

IPOL Journal

npm

NixOS

python

Software Heritage

- on disk: ~700 TB (uncompressed); as a graph ~20 B nodes, ~200 B edges
- the largest public source code archive in the world (and growing!)

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Security synergies and outlook

Use cases — product needs

e.g., for <https://archive.softwareheritage.org>

Browsing

- `ls`
- `git log` (Linux kernel: 800K+ commits)

Wayback machine

- `tarball`
- `git bundle` (Linux kernel: 7M+ nodes)

Provenance tracking

- commit provenance (one/all contexts) note: requires backtracking
- origin provenance (one/all contexts)


Note: we therefore need both the direct Merkle DAG graph and its **transposed**

For the sake of it

- local graph topology
- connected component size
 - enabling question to identify the best approach (e.g., scale-up v. scale-out) to conduct large-scale analyses
- any other emerging property

Software Engineering topics

- software provenance analysis at this scale is pretty much unexplored yet
- industry frontier: increase granularity down to the individual line of code
- replicate at this scale (famous) studies that have generally been conducted on (much) smaller version control system samples to confirm/refute their findings
- ...



How do you query the Software Heritage archive?
(on a budget)

Software Heritage Graph dataset

Use case: large scale analyses of the most comprehensive corpus on the development history of free/open source software.



Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli

The Software Heritage Graph Dataset: Public software development under one roof

MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE

preprint: <http://deb.li/swhmsr19>

Dataset

- Relational representation of the full graph as a set of tables
- Available as open data: <https://doi.org/10.5281/zenodo.2583978>
- Chosen as subject for the **MSR 2020 Mining Challenge**

Formats

- Local use: PostgreSQL dumps, or Apache Parquet files (~1 TiB each)
- Live usage: Amazon Athena (SQL-queriable), Azure Data Lake

Sample query — most frequent first commit words

```
1 SELECT COUNT(*) AS c, word FROM (  
2   SELECT LOWER(REGEXP_EXTRACT(FROM_UTF8(  
3     message), '^\\w+')) AS word FROM revision)  
4 WHERE word != ''  
5 GROUP BY word ORDER BY COUNT(*) DESC LIMIT 5;
```

Sample query — most frequent first commit words

```
1 SELECT COUNT(*) AS c, word FROM (  
2   SELECT LOWER(REGEXP_EXTRACT(FROM_UTF8(  
3     message), '^\\w+')) AS word FROM revision)  
4 WHERE word != ''  
5 GROUP BY word ORDER BY COUNT(*) DESC LIMIT 5;
```

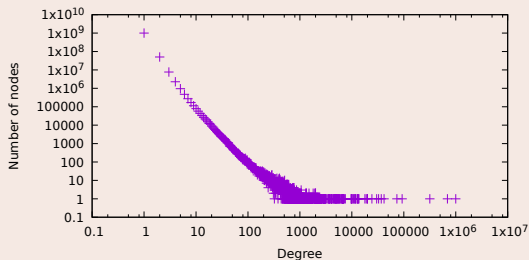
Count	Word
71 338 310	update
64 980 346	merge
56 854 372	add
44 971 954	added
33 222 056	fix

Sample query — fork and merge arities

Fork arity

i.e., how often is a commit based upon?

```
1 SELECT fork_deg, count(*) FROM (  
2   SELECT id, count(*) AS fork_deg  
3   FROM revision_history GROUP BY id) t  
4 GROUP BY fork_deg ORDER BY fork_deg;
```

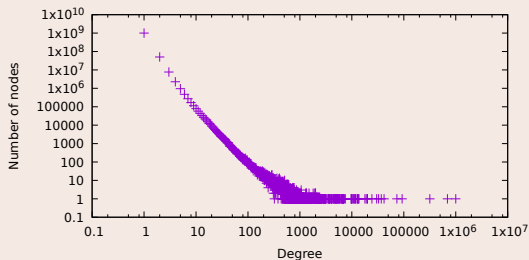


Sample query — fork and merge arities

Fork arity

i.e., how often is a commit based upon?

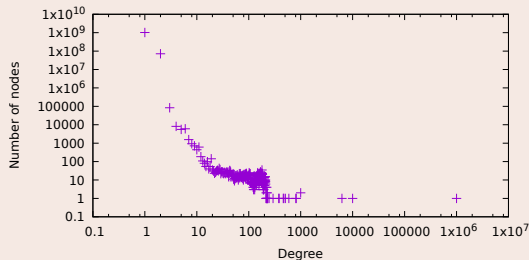
```
1 SELECT fork_deg, count(*) FROM (  
2   SELECT id, count(*) AS fork_deg  
3   FROM revision_history GROUP BY id) t  
4 GROUP BY fork_deg ORDER BY fork_deg;
```



Merge arity

i.e., how large are merges?

```
1 SELECT merge_deg, COUNT(*) FROM (  
2   SELECT parent_id, COUNT(*) AS merge_deg  
3   FROM revision_history GROUP BY parent_id  
4 GROUP BY deg ORDER BY deg;
```



Sample query — ratio of commits performed during weekends

```
1 WITH revision_date AS
2   (SELECT FROM_UNIXTIME(date / 1000000) AS date
3    FROM revision)
4 SELECT yearly_rev.year AS year,
5        CAST(yearly_weekend_rev.number AS DOUBLE)
6        / yearly_rev.number * 100.0 AS weekend_pc
7 FROM
8   (SELECT YEAR(date) AS year, COUNT(*) AS number
9    FROM revision_date
10   WHERE YEAR(date) BETWEEN 1971 AND 2018
11   GROUP BY YEAR(date) ) AS yearly_rev
12 JOIN
13   (SELECT YEAR(date) AS year, COUNT(*) AS number
14    FROM revision_date
15   WHERE DAY_OF_WEEK(date) >= 6
16         AND YEAR(date) BETWEEN 1971 AND 2018
17   GROUP BY YEAR(date) ) AS yearly_weekend_rev
18 ON yearly_rev.year = yearly_weekend_rev.year
19 ORDER BY year DESC;
```

Sample query — ratio of commits performed during weekends (cont.)

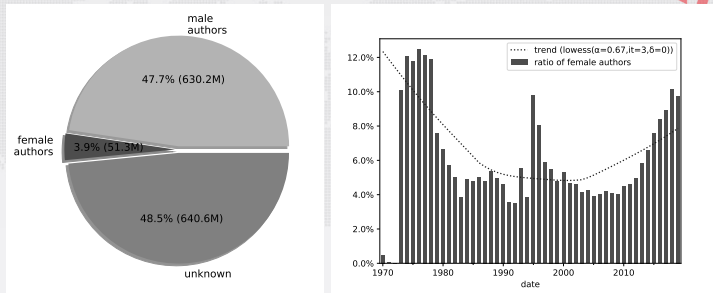
Year	Weekend	Total	Weekend percentage
2018	15130065	78539158	19.26
2017	33776451	168074276	20.09
2016	43890325	209442130	20.95
2015	35781159	166884920	21.44
2014	24591048	122341275	20.10
2013	17792778	88524430	20.09
2012	12794430	64516008	19.83
2011	9765190	48479321	20.14
2010	7766348	38561515	20.14
2009	6352253	31053219	20.45
2008	4568373	22474882	20.32
2007	3318881	16289632	20.37
2006	2597142	12224905	21.24
2005	2086697	9603804	21.72
2004	1752400	7948104	22.04
2003	1426022	6041502	20.54

Sample query — average size of the most popular file types

```
1 SELECT suffix,  
2     ROUND(COUNT(*) * 100 / 1e6) AS Million_files,  
3     ROUND(AVG(length) / 1024) AS Average_k_length  
4 FROM  
5     (SELECT length, suffix  
6      FROM -- File length in joinable form  
7           (SELECT TO_BASE64(sha1_git) AS sha1_git64, length  
8            FROM content ) AS content_length  
9      JOIN -- Sample of files with popular suffixes  
10         (SELECT target64, file_suffix_sample.suffix AS suffix  
11          FROM -- Popular suffixes  
12               (SELECT suffix FROM (  
13                  SELECT REGEXP_EXTRACT(FROM_UTF8(name),  
14                     '\.[^.]+$') AS suffix  
15                  FROM directory_entry_file) AS file_suffix  
16                  GROUP BY suffix  
17                  ORDER BY COUNT(*) DESC LIMIT 20 ) AS pop_suffix  
18          JOIN -- Sample of files and suffixes  
19               (SELECT TO_BASE64(target) AS target64,  
20                  REGEXP_EXTRACT(FROM_UTF8(name),  
21                     '\.[^.]+$') AS suffix  
22                  FROM directory_entry_file TABLESAMPLE BERNOULLI(1))  
23          AS file_suffix_sample  
24          ON file_suffix_sample.suffix = pop_suffix.suffix)
```

Sample study — 50 years of gender differences in code contributions

- start from the Software Heritage graph dataset
- detect gender of author names using standard tooling (gender-guesser)
- analyze both authors and commits over time, bucketing by commit timestamp



total commits by author gender (left), ratio of active female committers over time (right)



Stefano Zacchioli

Gender Differences in Public Code Contributions: a 50-year Perspective

IEEE Softw. 38(2): 45-50 (2021)

- one *can* query such a corpus SQL-style
- but relational representation shows its limits at this scale
 - ...at least as deployed on commercial SQL offerings such as Athena
- note: (naive) sharding is ineffective, due to the pseudo-random distribution of node identifiers
- experiments with Google BigQuery are ongoing
 - (we broke it at the first import attempt..., due to very large arrays in directory entry tables)

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Security synergies and outlook

 Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchiroli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering.
IEEE

Research question

Is it possible to efficiently perform software development history analyses at ultra large scale (= the scale of Software Heritage archive or more), on a single, relatively cheap machine?

Idea

Apply state-of-the-art graph compression techniques from the field of Web graph / social network analysis.

Background — (Web) graph compression

Definition (The graph of the Web)

Directed graph that has Web pages as nodes and hyperlinks between them as edges.

Properties (1)

- **Locality:** pages link to pages whose URLs are lexicographically similar. URLs share long common prefixes.

→ use **D-gap compression**

Adjacency lists

Node	Outdegree	Successors
...
15	11	13,15,16,17,18,19,23,24,203,315,1034
16	10	15,16,17,22,23,24,315,316,317,3041
17	0	
18	5	13,15,16,17,50
...

D-gapped adjacency lists

Node	Outdegree	Successors
...
15	11	3,1,0,0,0,3,0,178,111,718
16	10	1,0,0,4,0,0,290,0,0,2723
17	0	
18	5	9,1,0,0,32
...

Background — (Web) graph compression (cont.)

Definition (The graph of the Web)

Directed graph that has Web pages as nodes and hyperlinks between them as edges.

Properties (2)

- **Similarity:** pages that are close together in lexicographic order tend to have many common successors.

→ use **reference compression**

Adjacency lists

Node	Outd.	Successors
...
15	11	13,15,16,17,18,19,23,24,203,315,1034
16	10	15,16,17,22,23,24,315,316,317,3041
17	0	
18	5	13,15,16,17,50
...

Copy lists

Node	Ref.	Copy list	Extra nodes
...
15	0		13,15,16,17,18,19,23,24,203,315,1034
16	1	01110011010	22,316,317,3041
17			
18	3	11110000000	50
...	

Nodes

Node type	N. of nodes
origins	88 M
snapshots	57 M
releases	9.9 M
revisions	1.1 B
directories	4.9 B
contents	5.5 B
Total nodes	12 B

Edges

Edge type	N. of edges
origin → snapshot	195 M
snapshot → revision	616 M
snapshot → release	215 M
release → revision	9.9 M
revision → revision	1.2 B
revision → directory	1.1 B
directory → directory	48 B
directory → revisiony	482 M
directory → content	112 B
Total edges	165 B

Stats for archive snapshot 2018-09-25, from the Software Heritage graph dataset.

Growth rate: exponential, doubling every 22-30 months, cf.:

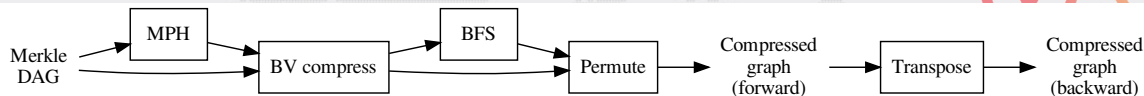


Roberto Di Cosmo, Guillaume Rousseau, Stefano Zacchiroli

Software Provenance Tracking at the Scale of Public Source Code

Empirical Software Engineering 25(4): 2930-2959 (2020)

Graph compression pipeline




- **MPH**: minimal perfect hash, mapping Merkle IDs to 0..N-1 integers
- **BV compress**: Boldi-Vigna compression (based on MPH order)
- **BFS**: breadth-first visit to renumber
- **Permute**: update BV compression according to BFS order

(Re)establishing locality

- key for good compression is a node ordering that ensures locality and similarity
- which is very much *not* the case with Merkle IDs, ...but is the case *again* after BFS reordering

Compression experiment



Step	Wall time (hours)
MPH	2
BV Compress	84
BFS	19
Permute	18
Transpose	15
Total	138 (6 days)

- server equipped with 24 CPUs and 750 GB of RAM
- RAM mostly used as I/O cache for the BFS step
- *minimum* memory requirements are close to the RAM needed to load the final compressed graph in memory

Compression efficiency (space)

Forward graph

total size	91 GiB
bits per edge	4.91
compression ratio	15.8%

Backward graph

total size	83 GiB
bits per edge	4.49
compression ratio	14.4%

Operating cost

The structure of a full bidirectional archive graph fits in less than 200 GiB of RAM, for a hardware cost of ~300 USD.

Compression efficiency (time)

Benchmark — Full BFS visit (single thread)

Forward graph

wall time	1h48m
throughput	1.81 M nodes/s (553 ns/node)

Backward graph

wall time	3h17m
throughput	988 M nodes/s (1.01 μ s/node)

Benchmark — Edge lookup

random sample: 1 B nodes (8.3% of entire graph); then enumeration of all successors

Forward graph

visited edges	13.6 B
throughput	12.0 M edges/s (83 ns/edge)

Backward graph

visited edges	13.6 B
throughput	9.45 M edges/s (106 ns/edge)

Note how edge lookup time is close to DRAM random access time (50-60 ns).

Incrementality

compression is **not incremental**, due to the use of contiguous integer ranges

- but the graph is append-only, so...
- ...based on expected graph growth rate it should be possible to pre-allocate enough free space in the integer ranges to support **amortized incrementality** (future work)

Incrementality

compression is **not incremental**, due to the use of contiguous integer ranges

- but the graph is append-only, so...
- ...based on expected graph growth rate it should be possible to pre-allocate enough free space in the integer ranges to support **amortized incrementality** (future work)

In-memory v. on-disk

the compressed in-memory graph structure has **no attributes**

- usual design is to exploit the $0..N-1$ integer ranges to **memory map node attributes** to disk for efficient access
- works well for queries that does graph traversal first and "join" node attributes last; ping-pong between the two is expensive
- edge attributes are more problematic (work in progress)

- 
- 1 Software Heritage
 - 2 Querying the archive
 - 3 Graph compression
 - 4 Security synergies and outlook

Securing the open source supply chain

Software supply chain attacks are becoming more and more popular and raising in profile. → Cf. *SolarWindws attacks* (2021), breaching several US govt. branches

Definition — Reproducible Builds (R-B)

The build process of a software product is **reproducible** if, after designating a specific version of its source code and all of its build dependencies, every build produces **bit-for-bit identical artifacts**, no matter the environment in which the build is performed.

- R-B allows to **increase trust in binary executables** built from trusted (open source) code by untrusted 3rd-party software vendors (e.g., app stores, distros)
- The **reproducible-builds.org project** has popularized the notion, is backed by major open source industry players, and has made large open source software collections reproducible (e.g., 95% of Debian packages)

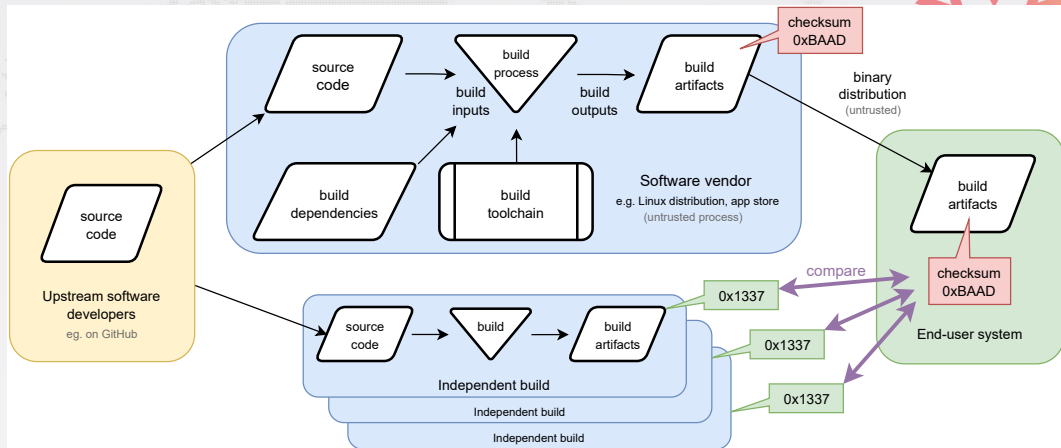


Chris Lamb, Stefano Zacchiroli

Reproducible Builds: Increasing the Integrity of Software Supply

IEEE Software 2021 (to appear, DOI 10.1109/MS.2021.3073045)

Securing the open source supply chain (cont.)



Securing the open source supply chain (cont.)

- Software Heritage provides key ingredients for R-B pipelines: on-demand archival (e.g., of VCS commits referenced by build recipes) + long-term availability
- We have implemented this by integrating the GNU Guix package manager with Software Heritage

Software Heritage and GNU Guix join forces
to enable long term reproducibility



Connecting reproducible deployment to a long-term source code archive



Ludovic Courès — March 29, 2019

GNU Guix can be used as a “package manager” to install and upgrade software packages as is familiar to GNU/Linux users, or as an environment manager, but it can also provision containers or virtual machines, and manage the operating system running on your machine.

One foundation that sets it apart from other tools in these areas is reproducibility. From a high-level view, Guix allows users to declare complete software environments and instantiate them. They can share those environments with others, who can replicate them or adapt them to their needs. This aspect is key to reproducible computational experiments: scientists need to reproduce software environments before they can reproduce experimental results, and this is one of the things we are focusing on in the context of the Guix-HPC effort. At a lower level, the project, along with others in the Reproducible Builds community, is working to ensure that software build outputs are reproducible, bit for bit.

Work on reproducibility at all levels has been making great progress. Guix, for instance, allows you to travel back in time. That Guix can travel back in time and build software reproducibly is a great step forward. But there's still an important piece that's missing to make this viable: a stable source code archive. This is where Software Heritage (SWH for short) comes in.

When source code vanishes

- <https://www.softwareheritage.org/2019/04/18/software-heritage-and-gnu-guix-join-forces-to-enable-long-term-reproducibility>
- <https://guix.gnu.org/blog/2019/connecting-reproducible-deployment-to-a-long-term-source-code-archive/>

Tracking of vulnerable source code artifacts

Software Heritage provides a unique observatory on the (best approximation of) the entire *Software Commons*, i.e., all software published in source code form

Software provenance tracking at the scale of the world

- by following the *transposed* Software Heritage graph we can locate **all known public occurrences** of source code artifacts (individual source files, entire source tree, commits) in other commits or repositories
- we have developed two approaches to do that:
 - 1 database-based (Rousseau et al. EMSE 2020): incremental, answers a fixed set of queries, requires significant disk space
 - 2 compressed-graph-based (Boldi et al. SANER 2020): non-incremental, flexible graph-base querying, fits in RAM
- current applications: "intellectual property"/prior art, open source license compliance, software composition analysis (SCA) → collab. with CAST

Tracking of vulnerable source code artifacts (cont.)

Adding in-memory commit timestamps (experimental)

Idea: in-memory timestamp array (us precision, 8 bytes each), indexed by revision node id. This enables to efficiently exploit timestamp information during graph visits.

Finding the *earliest* commit referencing a source file/dir

Early experiment: finding the earliest revision containing a given file using in-memory commit timestamps, on 10 M randomly selected blobs.

Mean lookup time: 4.1 ms (avg on 95% percentile: 2.2 ms)

Tracking vulnerable source code files/trees

Given a source file/tree affected by a known vulnerability (e.g., identified by a CVE) we can efficiently identify *all* commits (and repositories, extending the traversals) that reference it, triggering further inspection. Furthermore, we can efficiently select which commits to filter out during visits (e.g., "recent" ones, only in selected repos, etc.), based on timestamps of other attributes (that fit in memory or are mmap()-ed to disk).

v. State-of-the-art industry offerings

Similar to what GitHub/GitLab offer as a service, but:

- without having to rely on repository scanning, because the "big picture" is already present in the Software Heritage archive by design
- independent from the development platform vendor (e.g., a "vulnerable file" primarily hosted on GitHub can be spotted in GitLab repositories and vice-versa)
- complementary and synergistic with analyses of vulnerable dependency information (which are also available in Software Heritage via metadata mining)

Caveats

- current granularity stops at the file level and traceability breaks with even just whitespace changes. Increasing tracking granularity to the snippet/line of code level is possible, but untested at this scale yet (cf. research roadmap)

Graph compression

- incremental, amortized compression → ongoing UniMi collaboration
- graph query languages on top of the compressed representation → LIRIS collaboration (early stages)

A (brief) research roadmap — 1

Graph compression

- incremental, amortized compression → ongoing UniMi collaboration
- graph query languages on top of the compressed representation → LIRIS collaboration (early stages)

Complex networks

- local topology of the global VCS graph
- emergent properties (the "classics": scale-free, small world, etc.)
- dynamic modeling of graph evolution over time → collab. with physics @ UParis



Antoine Pietri, Guillaume Rousseau, Stefano Zacchioli

Determining the Intrinsic Structure of Public Software Development History

MSR 2020: 17th Intl. Conf. on Mining Software Repositories. IEEE

registered study protocol

A (brief) research roadmap — 2

Very-large-scale "big code"

- *big code* = apply ML/DL to source code and other development byproducts
- current results are language-specific and limited in scale; even the simplest problems become challenging at this scale and heterogeneity
 - lead: scalable language detection → collaboration with UniBo
 - lead: project classification → collaboration with CELI
- the VCS graph remains largely unexplored in big code
 - lead: use GNN for VCS node classification → ANR COREOGRAPHIE

A (brief) research roadmap — 2

Very-large-scale "big code"

- *big code* = apply ML/DL to source code and other development byproducts
- current results are language-specific and limited in scale; even the simplest problems become challenging at this scale and heterogeneity
 - lead: scalable language detection → collaboration with UniBo
 - lead: project classification → collaboration with CELI
- the VCS graph remains largely unexplored in big code
 - lead: use GNN for VCS node classification → ANR COREOGRAPHIE

Very-large-scale source code indexing

- common AST-based approaches for code indexing are not viable here due to maximum heterogeneity
- alternative: treat code as text and full-text index it
 - previous exp.: 3-gram based indexing in Debsources, supporting regexp matching
- goal: find a sweet spot between the two

Very-large-scale reproducibility in software engineering

- most results in empirical software engineering are determined on corpora significantly smaller than Software Heritage
→ external validity threat; do results generalize to the full body of public code?
- 2-year research plan
 - 1 identify impactful sw. eng. studies that can be reproduced using Software Heritage
 - selected topics (tentative): code reuse, code quality, project classification, technical debt, developer productivity
 - 2 reproduce selected studies one-by-one, at Software Heritage scale
 - 3 document findings, e.g., via RENE (REproducibility Studies and NEgative Results) scientific initiatives
- collaboration with Microsoft Research (just started)

Wrapping up

- Software Heritage archives all public source code as a huge Merkle DAG
- Querying and analyzing it at scale (20/200 B nodes/edges) is an open problem
- Gold mine of research leads in sw. eng., big code, reproducibility, security

References (selected)



Jean-François Abramatic, Roberto Di Cosmo, Stefano Zacchioli
Building the Universal Archive of Source Code
Communications of the ACM, October 2018



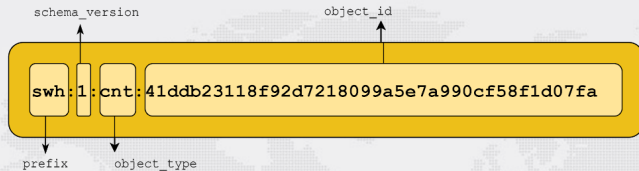
Antoine Pietri, Diomidis Spinellis, Stefano Zacchioli
The Software Heritage graph dataset: public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE

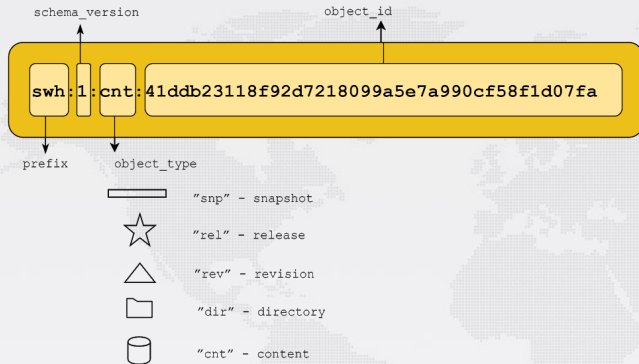


Paolo Boldi, Antoine Pietri, Sebastiano Vigna, Stefano Zacchioli
Ultra-Large-Scale Repository Analysis via Graph Compression
SANER 2020, 27th Intl. Conf. on Software Analysis, Evolution and Reengineering. IEEE

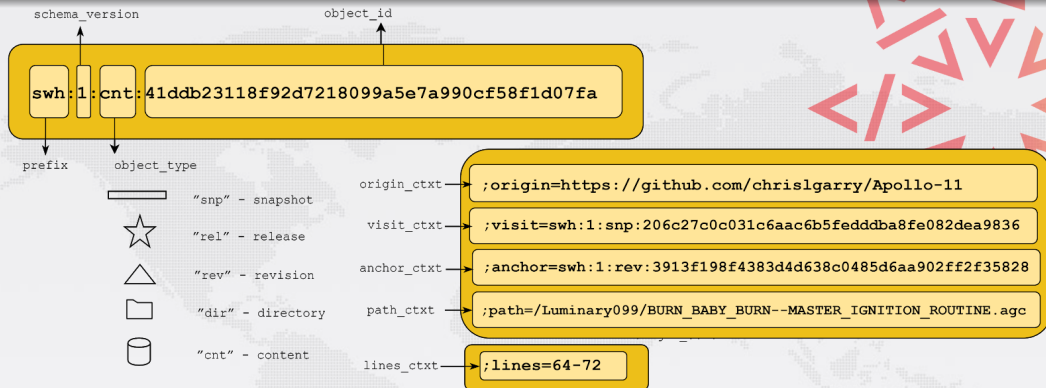
Contacts

Stefano Zacchioli / upsilon.cc / zack@upsilon.cc / [@zacchiro](https://twitter.com/zacchiro)



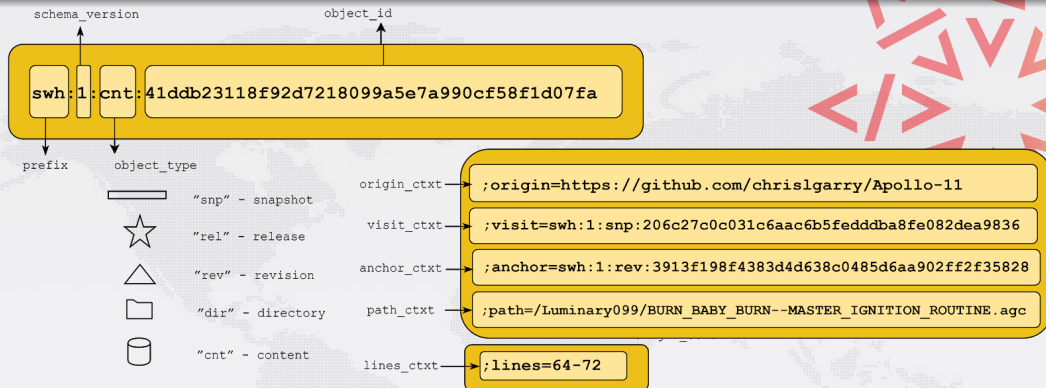






An emerging standard

- in Linux Foundation's SPDX 2.2
- IANA-registered "swh:" URI prefix
- WikiData property P6138



An emerging standard

- in Linux Foundation's SPDX 2.2
- IANA-registered "swh:" URI prefix
- WikiData property P6138

Examples

- Apollo 11 AGC excerpt
- Quake III rsqrt