Software Heritage

Analyzing the Global Graph of Public Software Development

Stefano Zacchiroli

Télécom Paris — zack@upsilon.cc, @zacchiro

14 Oct 2021

Team DIG — Télécom Paris

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

About me

- Professor, Télécom Paris, ACES team (newbie!)
- 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 interests

- Software engineering, Free/Open Source Software (FOSS)
- Digital commons
- Computer security
- Software supply chain

Outline

1. Software Heritage
2. Querying the archive
3. Graph compression
4. Research leads (in and around graph mining)

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

INVENTEURS DU MONDE NUMÉRIQUE

Platinum sponsors

Gold sponsors

Silver sponsors

Bronze sponsors

www.softwareheritage.org/support/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

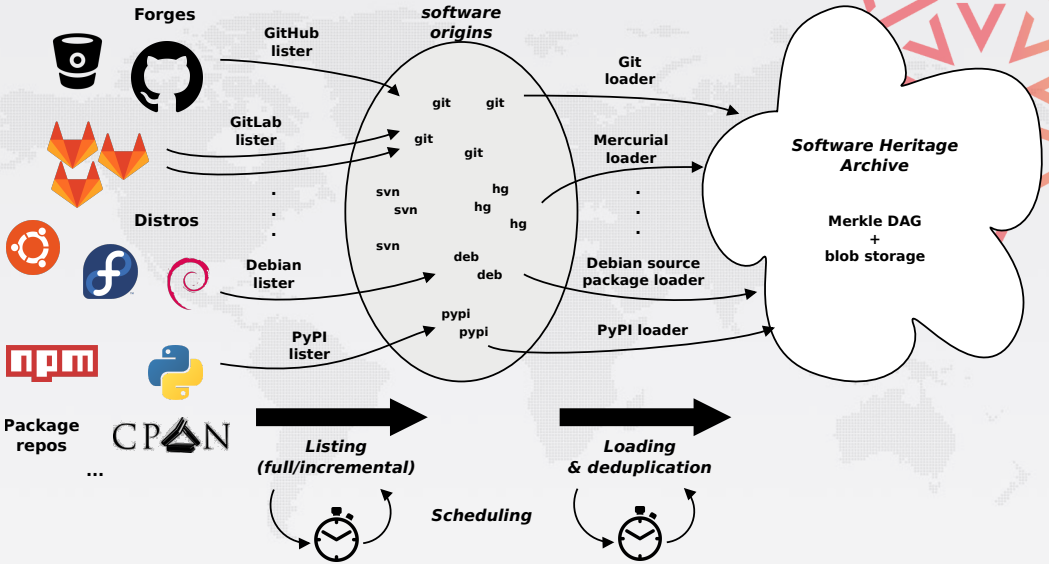
… in a VCS-/archive-agnostic canonical data model

We DON'T archive (yet)

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

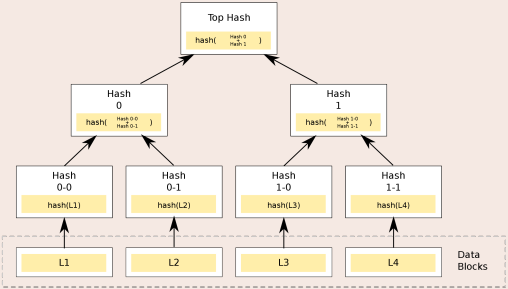
Long term vision: play our part in a *"semantic wikipedia of software"*

Data flow


Forges
GitHub lister
GitLab lister
Distros
Debian lister
PyPI lister
Package repos
CPAN
...
software origins
git
git
git
git
svn
svn
svn
hg
hg
hg
deb
deb
pypi
pypi
Git loader
Mercurial loader
Debian source package loader
PyPI loader
Software Heritage Archive
Merkle DAG
+
blob storage
Listing (full/incremental)
Loading & deduplication
Scheduling

Merkle trees

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

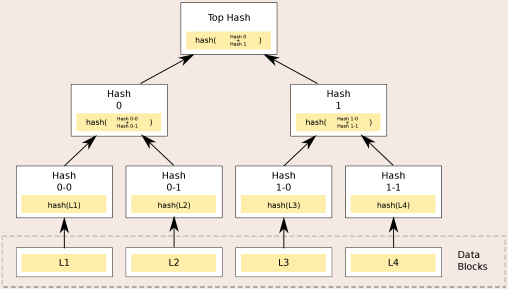


Combination of

- tree
- hash function

Merkle trees

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



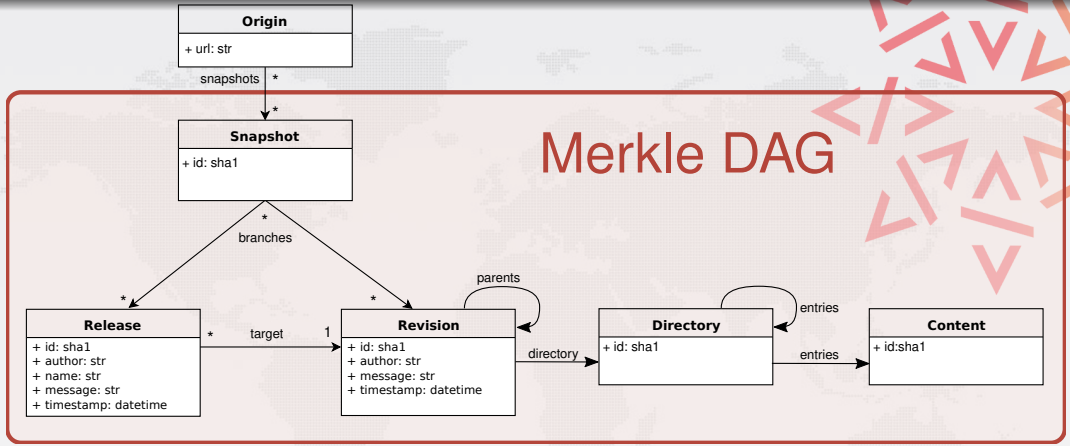
Combination of

- tree
- hash function

Classical cryptographic construction

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

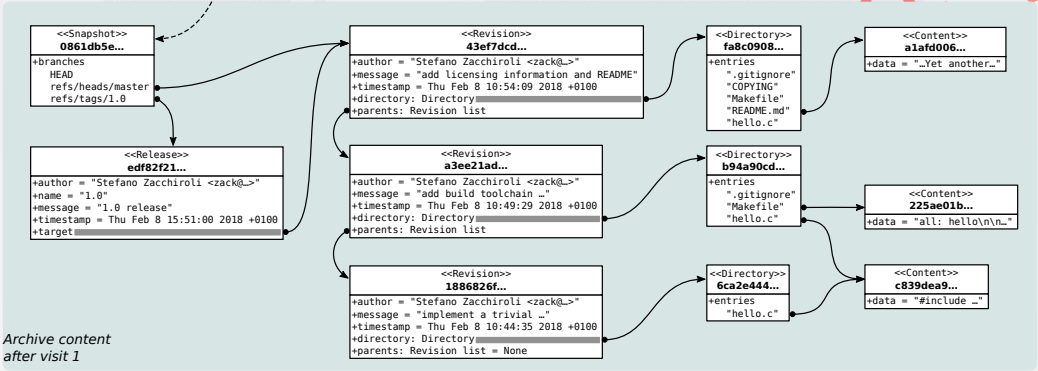
Data model



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

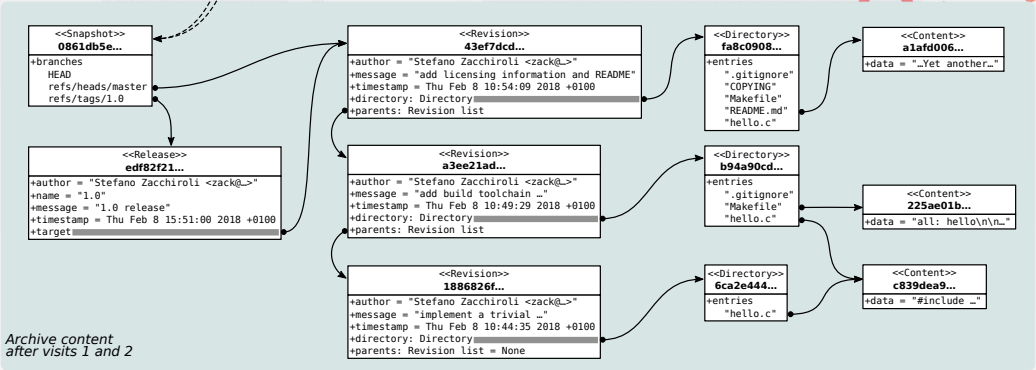
origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100



Archive content after visit 1

The archive: a (giant) Merkle DAG

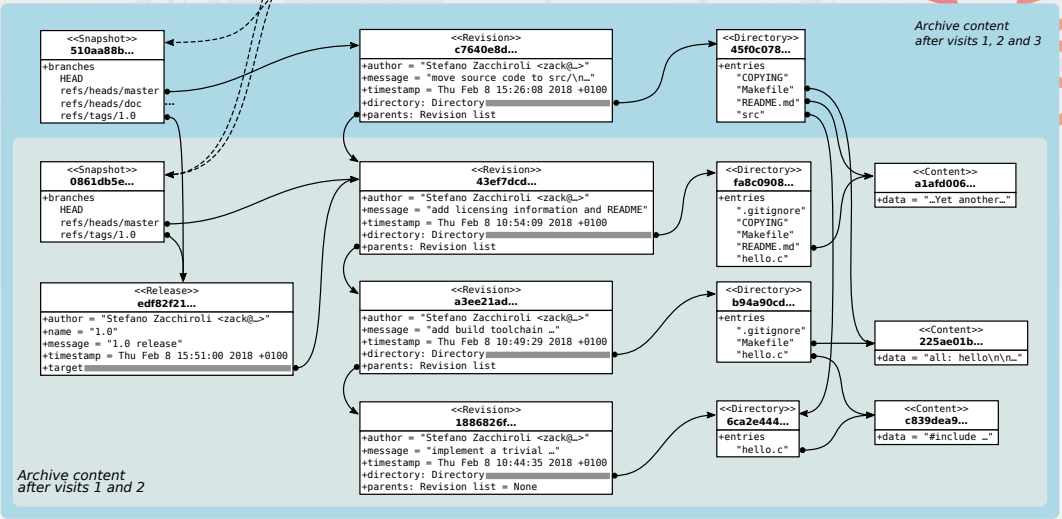
origin	visit	snapshot	timestamp
https://forge.softwareheritage.org/source/helloworld.git	1	0861db5e...	Fri Feb 9 12:38:45 2018 +0100
https://forge.softwareheritage.org/source/helloworld.git	2	0861db5e...	Fri Feb 9 13:29:00 2018 +0100



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



Archive coverage — archive.softwareheritage.org

Bitbucket · R · debian · FramaGit · GitHub · GitLab · GITORIOUS · Google code · GNU · Guix · HAL archives-ouvertes.fr · Inria · IPOL Journal · npm · NixOS · python

Archive coverage — archive.softwareheritage.org

Bitbucket · R · debian · Framagit · GitHub · GitLab · GITORIOUS · Google code · GNU · Guix · HAL archives-ouvertes.fr · Inria · IPOL Journal · npm · NixOS · python

- 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!)

Outline

1. Software Heritage
2. Querying the archive
3. Graph compression
4. Research leads (in and around graph mining)

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

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

Sample query — most frequent first commit words

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

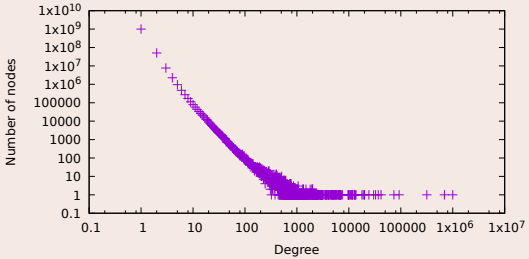
Count	Word
71 338 310	update
64 980 346	merge
56 854 372	add
44 971 954	added
33 222 056	fix

Sample query — fork and merge arities

Fork arity

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

```sql
SELECT fork_deg, count(*) FROM (
  SELECT id, count(*) AS fork_deg
  FROM revision_history GROUP BY id) t
GROUP BY fork_deg ORDER BY fork_deg;
```

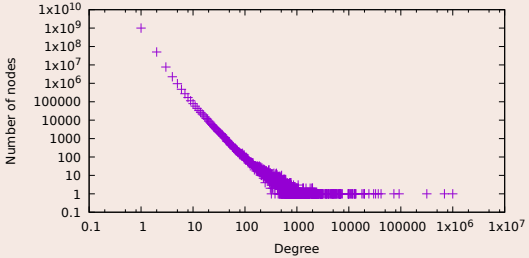


Sample query — fork and merge arities

Fork arity

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

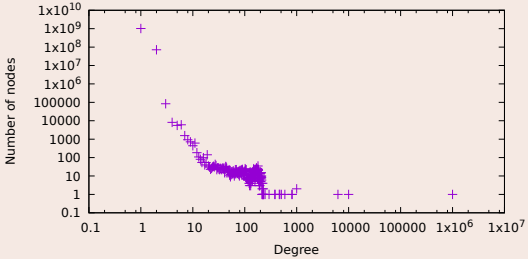
```
SELECT fork_deg, count(*) FROM (
  SELECT id, count(*) AS fork_deg
  FROM revision_history GROUP BY id) t
GROUP BY fork_deg ORDER BY fork_deg;
```



Merge arity

i.e., how large are merges?

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



Sample query — ratio of commits performed during weekends

```sql
WITH revision_date AS
  (SELECT FROM_UNIXTIME(date / 1000000) AS date
  FROM revision)
SELECT yearly_rev.year AS year,
  CAST(yearly_weekend_rev.number AS DOUBLE)
  / yearly_rev.number * 100.0 AS weekend_pc
FROM
  (SELECT YEAR(date) AS year, COUNT(*) AS number
  FROM revision_date
  WHERE YEAR(date) BETWEEN 1971 AND 2018
  GROUP BY  YEAR(date) ) AS yearly_rev
JOIN
  (SELECT YEAR(date) AS year, COUNT(*) AS number
  FROM revision_date
  WHERE DAY_OF_WEEK(date) >= 6
      AND YEAR(date) BETWEEN 1971 AND 2018
  GROUP BY  YEAR(date) ) AS yearly_weekend_rev
  ON yearly_rev.year = yearly_weekend_rev.year
ORDER BY  year DESC;
```

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

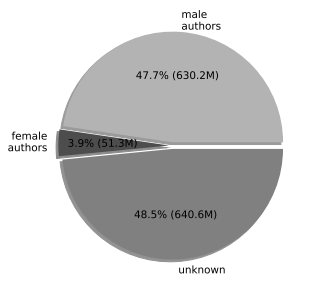
Year	Weekend	Total	Weekend percentage
2018	15130065	78539158	19.26
2017	33776451	168074276	20.09
2016	43890325	209442130	20.95
2015	35781159	166884920	21.44
2014	24591048	122341275	20.10
2013	17792778	88524430	20.09
2012	12794430	64516008	19.83
2011	9765190	48479321	20.14
2010	7766348	38561515	20.14
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	1426033	6941503	20.54

Sample query — average size of the most popular file types

```sql
SELECT suffix,
  ROUND(COUNT(*) * 100 / 1e6) AS Million_files,
  ROUND(AVG(length) / 1024) AS Average_k_length
FROM
  (SELECT length, suffix
  FROM  -- File length in joinable form
    (SELECT TO_BASE64(sha1_git) AS sha1_git64, length
    FROM content ) AS content_length
  JOIN  -- Sample of files with popular suffixes
  (SELECT target64, file_suffix_sample.suffix AS suffix
  FROM  -- Popular suffixes
    (SELECT suffix FROM (
      SELECT REGEXP_EXTRACT(FROM_UTF8(name),
       '\.[^.]+$') AS suffix
    FROM directory_entry_file) AS file_suffix
    GROUP BY  suffix
    ORDER BY  COUNT(*) DESC LIMIT 20 ) AS pop_suffix
  JOIN  -- Sample of files and suffixes
    (SELECT TO_BASE64(target) AS target64,
      REGEXP_EXTRACT(FROM_UTF8(name),
        '\.[^.]+$') AS suffix
    FROM directory_entry_file TABLESAMPLE BERNOULLI(1))
    AS file_suffix_sample
  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 commiters over time (right)

Stefano Zacchiroli
Gender Differences in Public Code Contributions: a 50-year Perspective
IEEE Softw. 38(2): 45-50 (2021)

Discussion

- 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)

Outline

1. Software Heritage
2. Querying the archive
3. Graph compression
4. Research leads (in and around graph mining)

Graph compression on the Software Heritage archive

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	3,1,0,0,0,0,3,0,178,111,718
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
…	…	…	

Corpus

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

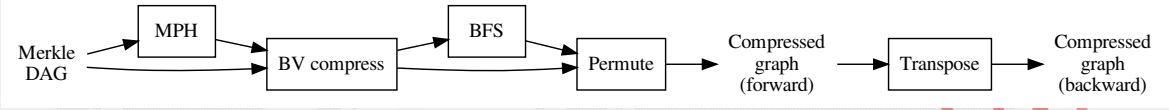
Edges

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 µs/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).

Discussion

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)

Discussion

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)

Outline

1. Software Heritage
2. Querying the archive
3. Graph compression
4. Research leads (in and around graph mining)

Research leads — Graph compression and querying

Graph compression

- Incremental, amortized compression

Graph querying

- Port (via compilation or dedicated runtime) graph query languages (e.g., TinkerPop, Cypher) to compressed graph representations

Research leads — Graph topology and visualization

Research leads — Graph topology and visualization

Complex networks

- Local topology of the global VCS graph

Antoine Pietri, Guillaume Rousseau, Stefano Zacchiroli
Determining the Intrinsic Structure of Public Software Development History
MSR 2020: 17th Intl. Conf. on Mining Software Repositories. IEEE

- Emergent properties (the "classics": scale-free, small world, etc.)
- Dynamic modeling of graph evolution over time
- Same for selected overlay graphs (e.g., collaboration among FOSS developers)

Research leads — Graph topology and visualization

Complex networks

- Local topology of the global VCS graph

Antoine Pietri, Guillaume Rousseau, Stefano Zacchiroli
Determining the Intrinsic Structure of Public Software Development History
MSR 2020: 17th Intl. Conf. on Mining Software Repositories. IEEE

- Emergent properties (the "classics": scale-free, small world, etc.)
- Dynamic modeling of graph evolution over time
- Same for selected overlay graphs (e.g., collaboration among FOSS developers)

Graph visualization

How do you visualize the Software Heritage graph? Use cases:

1. Give an intuitive sense of its topology (like the Web's "bow-tie structure")
2. Provide a "map of the stars" of FOSS development, for specific overlay graphs

State-of-the-art methods for automated graph visualization do not scale to this size.

Research leads — Big code

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
- VCS graphs, especially at this scale, remain largely unexplored in big code
  - Lead: use GNN and/or joint graph embeddings to classify VCS artifacts like individual commits and entire projects

Research leads — Big code

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
- VCS graphs, especially at this scale, remain largely unexplored in big code
  - Lead: use GNN and/or joint graph embeddings to classify VCS artifacts like individual commits and entire projects

Very-large-scale source code indexing

- AST-based approaches are not applicable due to maximum heterogeneity
- Alternative: treat code as text and full-text index it
  - Previous exp.: 3-gram based indexing of all Debian releases, supporting regexs
- Goal: find a sweet spot between the two

Research leads — Programming language identification

Recent results on (visual) programming language identification

- Francesca Del Bonifro, Maurizio Gabbrielli, Stefano Zacchiroli
  Content-Based Textual File Type Detection at Scale
  ICMLC 2021: The 13th International Conference on Machine Learning and Computing. ACM, 2021
- Francesca Del Bonifro, Maurizio Gabbrielli, Antonio Lategano, Stefano Zacchiroli
  Image-based many-language programming language identification
  In PeerJ Computer Science, 7:e631. PeerJ. 2021

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 and around large-scale graph mining

References (selected)

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

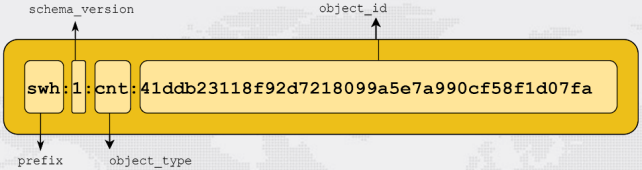
Antoine Pietri, Diomidis Spinellis, Stefano Zacchiroli
The Software Heritage graph dataset: public software development under one roof
MSR 2019: 16th Intl. Conf. on Mining Software Repositories. IEEE

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

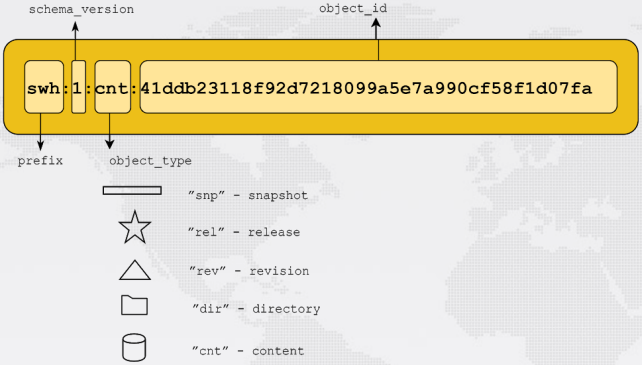
Contacts

Stefano Zacchiroli / upsilon.cc / zack@upsilon.cc / @zacchiro

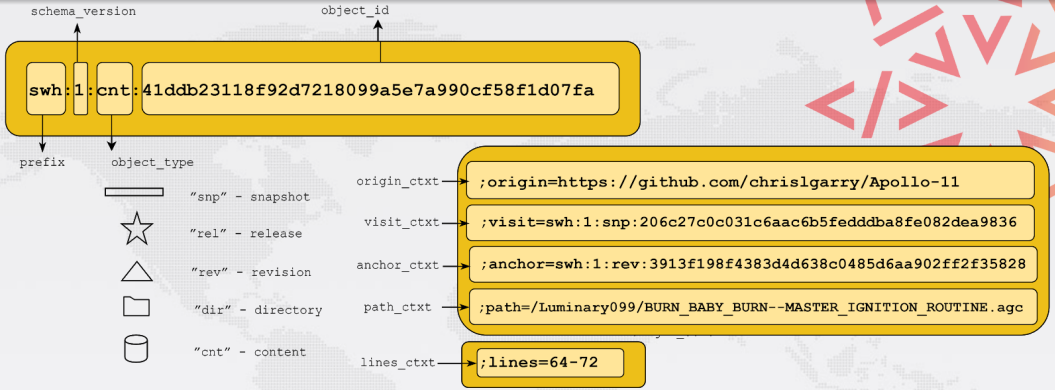
Meet the Software Heritage Identifiers (SWHIDs) (full spec)



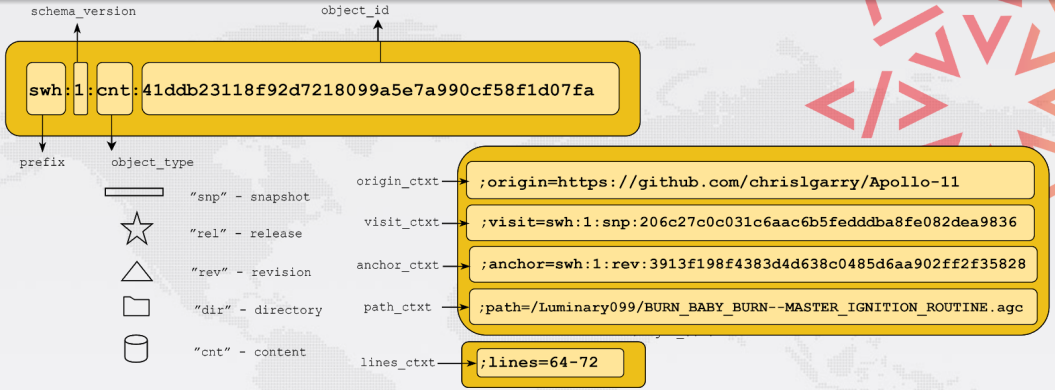
Meet the Software Heritage Identifiers (SWHIDs) (full spec)



Meet the Software Heritage Identifiers (SWHIDs) (full spec)



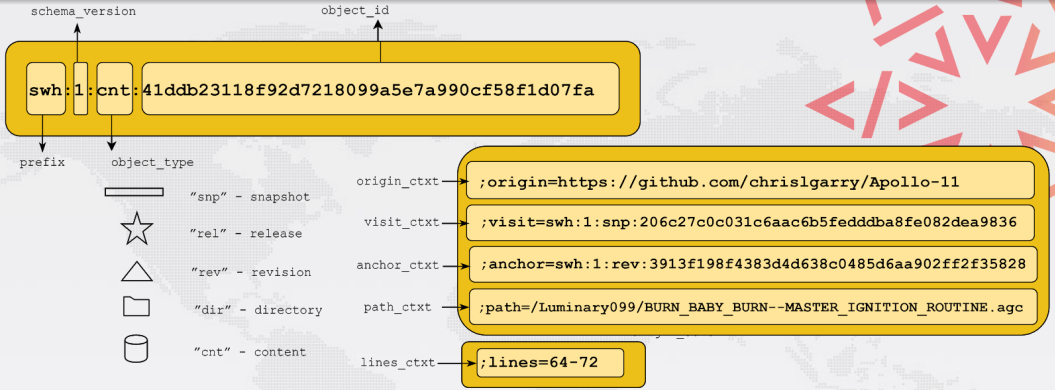
Meet the Software Heritage Identifiers (SWHIDs) (full spec)



An emerging standard

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

Meet the Software Heritage Identifiers (SWHIDs) (full spec)



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