Beyond FAIR Data: Making Research Software a First-Class Research Output

Roberto Di Cosmo

Director, Software Heritage
Inria and Université Paris Cité

8 September 2026

Software Heritage

THE GREAT LIBRARY OF SOURCE CODE

Outline

Short Bio: Roberto Di Cosmo

Computer Science professor in Paris, now working at INRIA

- *35+ years* of research (Theor. CS, Programming, Software Engineering, Erdos #: 3)
- *25+ years* of Free and Open Source Software
- *15+ years* building and directing structures for the common good

1999 *DemoLinux* – first live GNU/Linux distro

2007 *Free Software Thematic Group*
150 members 40 projects 200Me

2008 *Mancoosi project* www.mancoosi.org

2010 *IRILL* www.irill.org

2015 *Software Heritage* at INRIA

2018 *National Committee for Open Science*, France

2021 *EOSC Task Force on Infrastructures for Software*, European Union

COMMENT | 07 October 2025

Stop treating code like an afterthought: record, share and value it

Scientists, research institutions, funders, libraries and publishers must all improve software practices.

By Roberto Di Cosmo, Sabrina Granger, Konrad Hinsen, Nicolas Jullien, Daniel Le Berre, Yvolaine Louvet, Camille Maumet, Clémentine Maurice, Raphaël Monat & Nicolas P. Rougier

Illustration: Phil Wheeler

A *Nature* call to action

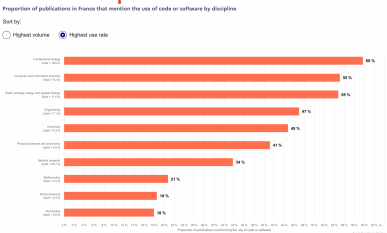
Nature commentary, October 2025

doi:10.1038/d41586-025-03196-0

This talk is the *how-to* behind that call: how to archive, reference, describe, cite and credit research software — concretely, and today.

Software is a pillar of Open Science

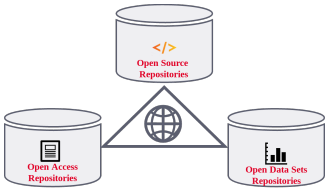
Software powers modern research



Over 20% of articles using software across all disciplines share it

2024 French Open Science Monitor

Key pillar: software



Links are important

Nota Bene

software may be a *tool*, a *research outcome* and a *research object*

access to the *source code* is essential!

Preserving (the history of) source code is necessary for *reproducibility*

Software *Source Code* is Precious Knowledge

Harold Abelson, Structure and Interpretation of Computer Programs (1st ed.) 1985

"Programs must be written for people to read, and only incidentally for machines to execute."

Apollo 11 source code (excerpt)

```
P63SPOT3        CA      BIT6            # IS THE LR ANTENNA IN POSITION 1 YET
                EXTEND
                RAND    CHAN33
                EXTEND
                BZF     P63SPOT4        # BRANCH IF ANTENNA ALREADY IN POSITION 1

                CAF     CODE500         # ASTRONAUT:    PLEASE CRANK THE
                TC      BANKCALL        #               SILLY THING AROUND
                CADR    GOPERF1
                TCF     GOTOPOOH        # TERMINATE
                TCF     P63SPOT3        # PROCEED       SEE IF HE'S LYING

P63SPOT4        TC      BANKCALL        # ENTER         INITIALIZE LANDING RADAR
                CADR    SETPOS1

                TC      POSTJUMP        # OFF TO SEE THE WIZARD ...
                CADR    BURNBABY
```

Quake III source code (excerpt)

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y; // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
// y = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this
can be removed

    return y;
}
```

Len Shustek, Computer History Museum 2006

"Source code provides a view into the mind of the designer."

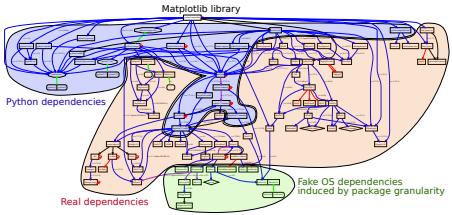
Software source code is *not* data (and FAIR is not the answer)

Software *evolves* over time

- projects may last decades
- the *development history* is key to its *understanding*

Complexity

- *millions* of lines of code
- large *web of dependencies*
  - easy to break, difficult to maintain
  - *research software* a thin top layer
- sophisticated *developer communities*



The human side

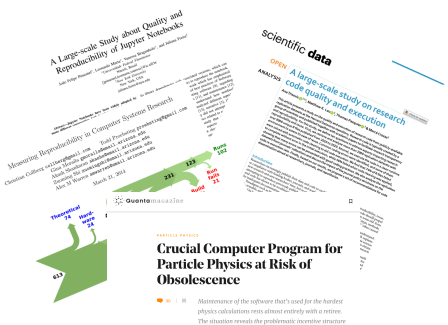
design, algorithm, code, test, documentation, community, funding

and so many more facets …

Outline

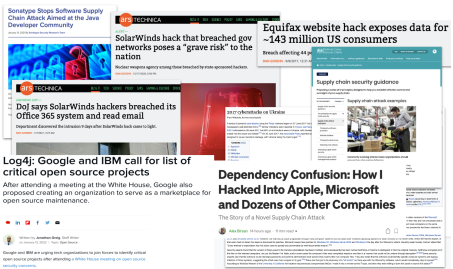
How are we managing our software ?

Reproducibility, maintenance in Academia



(articles: here, here, here and here)

Security, integrity, traceability in Industry



Can they track the software that they

- ship, use, acquire
- has that bug or vulnerability

Collberg's report from the trenches

Analysis of 613 papers

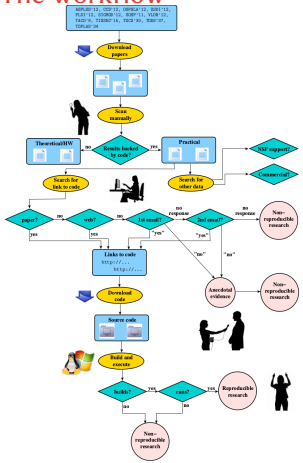
- 8 ACM conferences: ASPLOS'12, CCS'12, OOPSLA'12, OSDI'12, PLDI'12, SIGMOD'12, SOSP'11, VLDB'12
- 5 journals: TACO'9, TISSEC'15, TOCS'30, TODS'37, TOPLAS'34

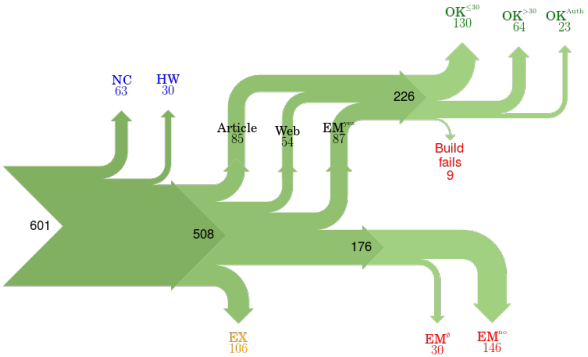
all very practical oriented

The basic question

can we get the code to build and run?

The workflow





...

... that's a whopping 40% of non reproducible works!

Pressure to make research code available is now raising

Evaluation of software artefacts — from optional contest to standard practice

- Started as contest (ESEC/FSE 2011, winner *Vouillon and Di Cosmo*)
- Now a near-universal standard: ACM Artifact Review and Badging v1.1 (5 badges, *left*)

Adopted across computer science communities:

- PL/SE: POPL, PLDI, OOPSLA, ICFP, ECOOP, ICSE, FSE, MSR, ISSTA, CAV
- Systems: OSDI, SOSP, NSDI, EuroSys, ASPLOS, MICRO, ISCA, HPCA
- Security: USENIX Security (availability mandatory in 2025), CCS, NDSS, IEEE S/P
- HPC and DB: SC (Repro. Initiative), SIGMOD (ARI), VLDB

Best-of-breed exemplars:

- IPOL: peer-reviewed papers ship code + runnable in-browser demo (e.g. non-local-means denoising)
- Graphics Replicability Stamp: 478 graphics papers independently reproduced

EuroSys 2025: 93% of papers earned "Functional"; ~47% pursued "Results Reproduced"

… and spreading beyond computer science

Economics and social sciences

- AEA Data Editor (Vilhuber, Cornell): mandatory code+data audit for all AEA journals (AER, AEJ family), since 2019
- cascad: non-profit certification agency (CNRS, HEC Paris, GENES, X); warrants numerical reproducibility, 170+ certifications

Life sciences, geography, and cross-disciplinary

- CodeCheck (Eglen, Nüst): independent execution of paper code; partner journals incl. GigaScience, Lifecycle Journal, Reproducible AGILE
- ReScience C: open peer-reviewed journal dedicated to replications of computational research

A pan-disciplinary standard

- NISO RP-31:2021: extends ACM-style badging to all sciences

Outline

Archive, Reference, Describe, Cite and Credit

Archive

Research software artifacts must be properly archived

make sure we can *retrieve* them (*reproducibility*)

Reference

Research software artifacts must be properly referenced

make sure we can *identify* them (*reproducibility*)

Describe

Research software artifacts must be properly described

make it easy to *discover* and *reuse* them (*visibility*)

Cite/Credit

Research software artifacts must be properly cited *(not the same as referenced!)*

to give *credit* to authors (*evaluation*!)

Archive and reference: some popular approaches that do not fit the bill

A - Since the ~~1970's~~ 1990's

.zip or .tar file on:

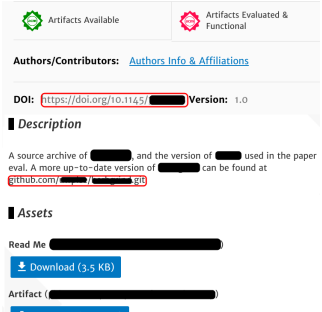
- ~~ftp server~~ (e.g. gnu)
- web page (example)
- document archive (+ DOI sample)

B - Since the 2000's

Rely on *software forges*

- institutional/project (e.g. example)
- free commercial ones: BitBucket, GitHub, GitLab, … (e.g. parmap)

C: a mix of the two



Can get no satisfaction…

A *Poor user experience*

B *No preservation guarantee*

C Can do *so much* better

Forges are *not* archives!

2015: the first big bad news

Google Code and Gitorious.org shutdown: ~1M endangered repositories

- broken links in the web of knowledge (my papers too)

Big bad news keep coming in

- summer 2019: BitBucket announces Mercurial VCS sunset
- july 2020: BitBucket erases *250.000+* repositories (including research software)
- summer 2022: GitLab.com considers erasing all projects that are inactive for a year

In Academia too!

- 2021: Inria's old gforge is unplugged… breaks the Opam build chain for OCaml

We need a universal archive of software source code: now we have one!

Outline

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 and share all software source code

Research infrastructure

enable analysis of all software source code

An international, non profit initiative built for the long term

Sharing the vision

And many more ...
www.softwareheritage.org/support/testimonials

Donors, members, sponsors

Inria

Diamond sponsors: Microsoft, TII Technology Innovation Institute

Platinum sponsors: cnrs, Ministères de l'Enseignement supérieur et de la Recherche, Infosys, HUAWEI, CEA

Gold sponsors: aws, Google, GitHub, openinventionnetwork, Université Paris Cité

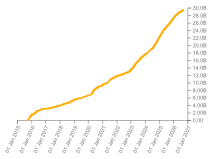
Silver sponsors: AdaCore, Premier Ministre, Direction interministérielle du numérique, Sorbonne Université, Università di Pisa

Bronze sponsors: NVIDIA, Sant'Anna, Scuola Normale Superiore, Université de Lorraine, université PARIS-SACLAY

The largest software archive, a shared infrastructure

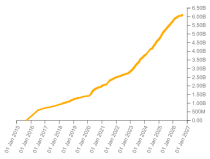
Source files

29,523,977,803



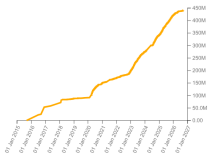
Commits

6,112,189,339



Projects

441,196,029



Directories

23,007,987,747

Authors

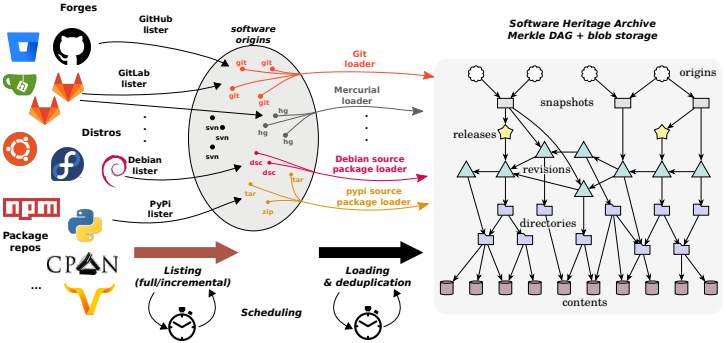
107,736,452

Releases

184,798,617

Bitbucket	git	R
1,925,997 origins	21,603 origins	21,113 origins
debian		GitHub
128,719 origins	5,947 origins	137,564,899 origins
GitLab	Guix	GNU
3,982,586 origins	12,032 origins	354 origins
heptapod	launchpad	Maven
1,068 origins	329,908 origins	93,738 origins
NixOS	npm	
12,032 origins	1,799,296 origins	4,083 origins
Phabricator	python	SOURCEFORGE
192 origins	410,582 origins	308,990 origins

Address common Open Science and Open Source needs: archival



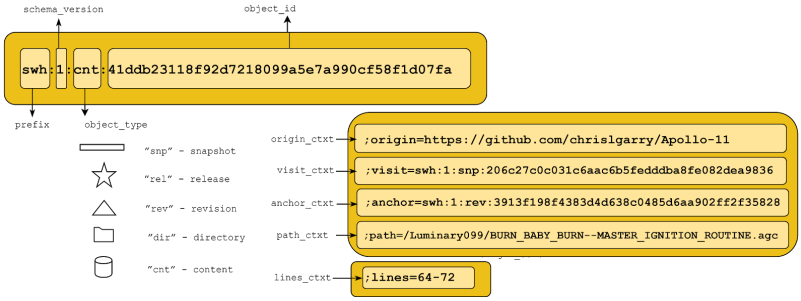
Global development history permanently archived in a uniform data model

- over 29.5 billion unique source files from over 441 million software projects
- ~3PB (compressed) blobs, ~60 B nodes, over 1 trillion edges

Meet the SWHID identifier

Software Hash Identifiers (SWHID)

link to full docs



25+B
intrinsic,
decentralised,
cryptographic

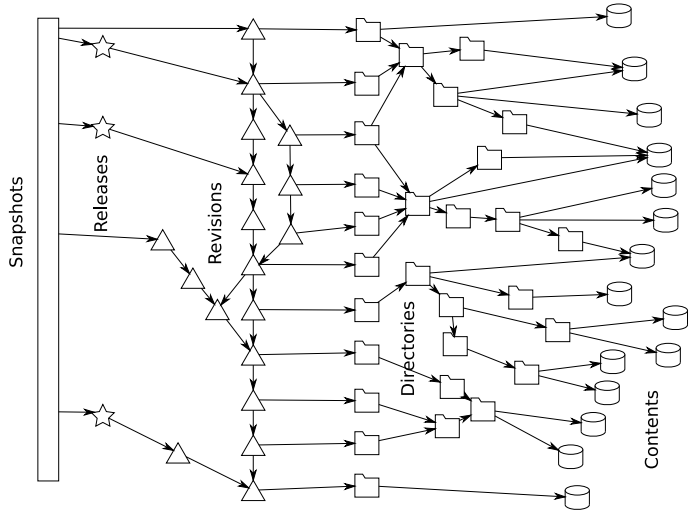
Full fledged *source code references* for traceability, integrity and reproducibility

- Linux Foundation SPDX 2.2
- IANA-registered "swh:"
- WikiData property P6138

Examples: Apollo 11 AGC excerpt, Quake III rsqrt
Guidelines available, see the HOWTO

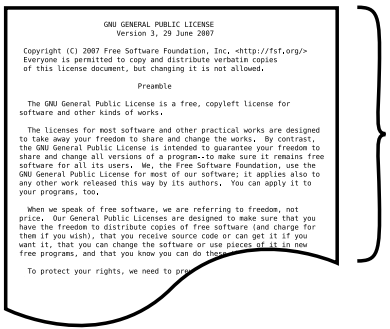
Now an ISO standard: ISO/IEC 18670:2025 ref.
implementation: swh.model

How a SWHID is built: a Merkle DAG of the development history



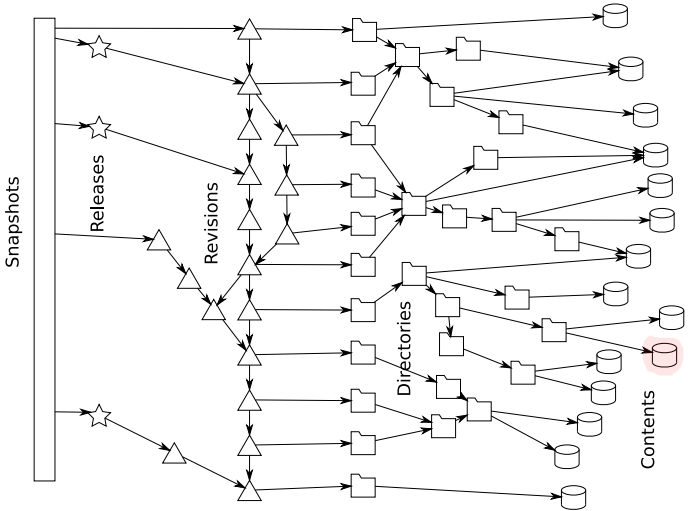
How a SWHID is built: a Merkle DAG of the development history

Contents



sha1: 8624bcdae55baeef...
sha256: 8ceb4b9ee5aded...
sha1_git: 94a9ed024d385...
length: 35147

How a SWHID is built: a Merkle DAG of the development history



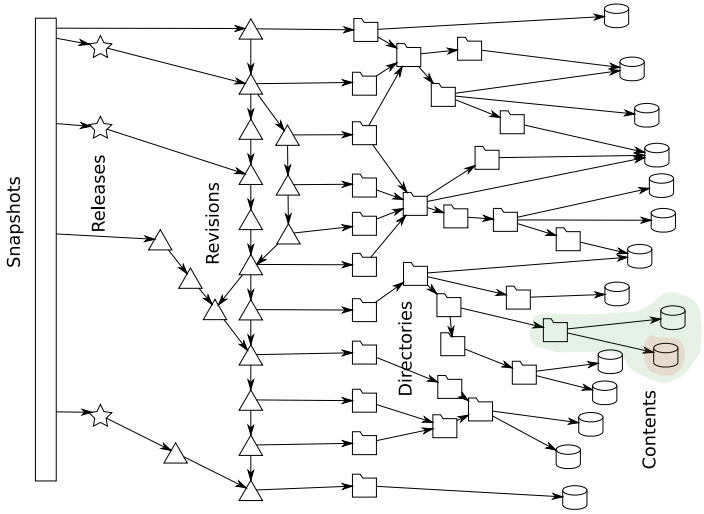
How a SWHID is built: a Merkle DAG of the development history

Directories

```
100644 blob c5baade4c44766042186ef858c0fd63d587ebf09 .gitignore
100644 blob 2d0a34af6f52cf3cf6b0c2f7bd0648fbd255e77f AUTHORS
100644 blob 94a9ed024d3859793618152ea559a168bbcbb5e2 LICENSE
100644 blob d9b2665a435a43f8a79a84e0867751dfb095c7bb MANIFEST.in
100644 blob 524175c2bad0b35b975f79284c2f5a6d5eaf2eb4 Makefile
100644 blob 5c7e3a5bbddb038682ba7793f440492ed9678bb3 Makefile.local
100644 blob 8617980629cd24e6080404f09aa749b085b3e07b README.db_testing
100644 blob 76b29f94cf815e0869c414d38d78d7ce08ec514e README.dev
040000 tree e1e10ecef948af0b93adb0372afc89f12e92618a bin
040000 tree 83e56d0beaf7793c77a45a345c80fcb8af503013 debian
040000 tree a34c9c4ba213f0cedc67f9816348d27955577af5 docs
100644 blob f2a6d32c6135aa7287bbd76167b01df2ae4f1539 requirements.txt
100755 blob eee147c36caf1bbc2d820da8dc026cb5b68180bc setup.py
040000 tree 224bb4c1f4c67fca1d160bffd2d06094e7e1abf3 sql
040000 tree 8631c9cd77bbe993168107ab5baf51f40c6300be swh
040000 tree 8fb905b56ba8ed692f1209b2773b474c6c1d66c1 utils
```

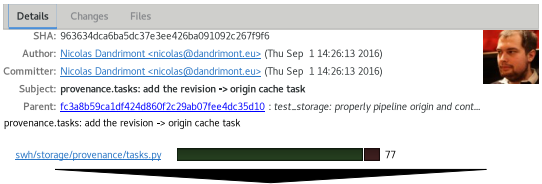
id: 515f00d44e92c65322aaa9bf3fa097c00ddb9c7d

How a SWHID is built: a Merkle DAG of the development history



How a SWHID is built: a Merkle DAG of the development history

Revisions

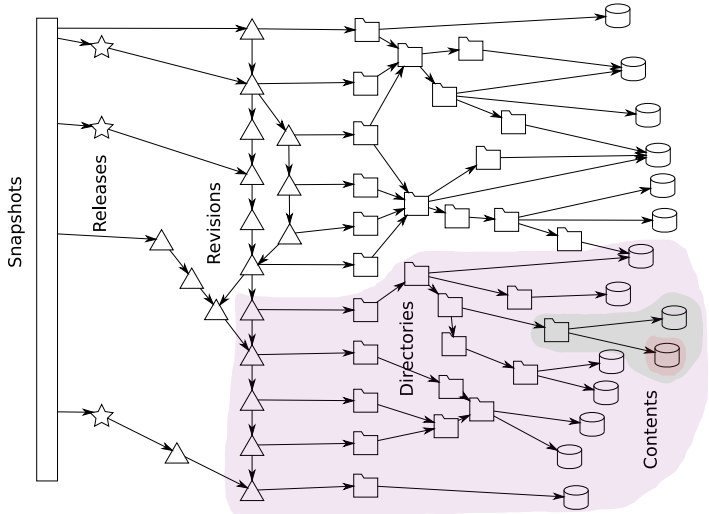


tree 515f00d44e92c65322aaa9bf3fa097c00ddb9c7d
parent fc3a8b59ca1df424d860f2c29ab07fee4dc35d10
author Nicolas Dandrimont <nicolas@dandrimont.eu> 1472732773 +0200
committer Nicolas Dandrimont <nicolas@dandrimont.eu> 1472732773 +0200

provenance.tasks: add the revision -> origin cache task

id: 963634dca6ba5dc37e3ee426ba091092c267f9f6

How a SWHID is built: a Merkle DAG of the development history



How a SWHID is built: a Merkle DAG of the development history

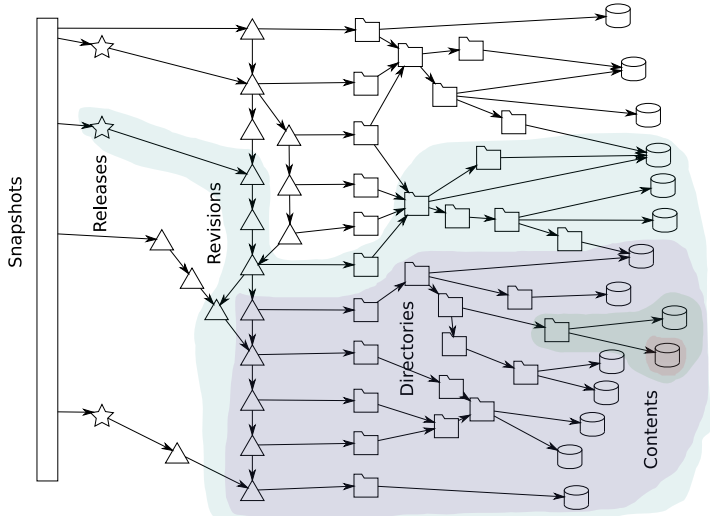
Releases

```
tag v0.0.51
Tagger: Nicolas Dandrimont <nicolas@dandrimont.eu>
Date:   Wed Aug 24 14:36:03 2016 +0200

Release swh,storage v0.0.51

 - Add new metadata column to origin_visit
 - Update swh-add-directory script for updated API
[...]

commit c0c9f16b1e134f593e7567570a1761b156e6eb1d
```

```
object c0c9f16b1e134f593e7567570a1761b156e6eb1d
type commit
tag v0.0.51
tagger Nicolas Dandrimont <nicolas@dandrimont.eu> 1472042163 +0200

Release swh,storage v0.0.51

 - Add new metadata column to origin_visit
 - Update swh-add-directory script for updated API
-----BEGIN PGP SIGNATURE-----

iQIzBAABCAAdBQJXvZTNFhxuaWNvbGFzQGRhbmRyaW1vbnQuZXUACgkQ7AWLMo2+
neqorw//aq6SOb5DijzEa+kWN3rXqVS+1K1vEVh1wNKAwx8eKJ7aX2kEiLDtt7uf
ahpZ6pz3q8nqs6aC1+YrxBfcih3L2YtrdZeWXWqr8xWNMaEoYDb8qaphwh8AD5t2
ICBIit2ujtXuCrDt93eKKPwvzZXg+hB0sMWy35Dr6jW7Z7K4Mu/PGglyIHPY55yo
lGEndWno7VfH1Vm6t1n5qB7l5mXRaqA+becqddubTZ2xjj+jpIUqC8cyqN3hm/fL
qsj2mu8kyz3t8tG/H1/pV+I5OwBlnPoS5TH0tujoJEVgPK/dHSP79QuHDHZFkCao
klJ6kAWyU80Mxb+nKV/joLbrR3+yWBFj3Qp5a1/V8oOTh6E1dALcNMpEaKCoKtMt
d/gMRax1I1/g0EDfnsW67G6sDwKPKPHhgfVLQ3nV3GaQQTnu1RpMz006H9/tAwzC
Gg/K1PdHT4hzOil46wYPZyje0U2VXGFu6vVU9vFQ4ZR/Wjn+0zMzdcRdrIJSUOMn
RpTTfUsbXUeXHGOpkgXhSYTnvp1gdPc76US5k0aGe84AZm1Ik0mGrwXCVfPqlYo
nhhibBSHBNMoqyF6yT5OpUbYk70tpYRRUGkWDeRK0wKSxkWkUZGtKzy6jYqJjo29
gulwgZQif5qWQCB0OontAL2+HvPFaVyckMejUhg62cP/+EHlvUk=
=kOxP
-----END PGP SIGNATURE-----
```

id: 85083a5cc14a441c89dea73f5bdf67c3f9c6afdb

How a SWHID is built: a Merkle DAG of the development history



How a SWHID is built: a Merkle DAG of the development history

git show-refs

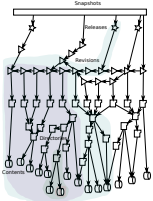
Snapshots

```
commit 08ffeb25770109525eb3ce21691466c53a1d9158 refs/heads/atime
commit ba5443a24e3f9fe323a46c292cec4fcbe61c67eb refs/heads/directory-listing-arrays
commit d69e0dbf892383ff6589b27fbe1c05d27238d9c5 refs/heads/foo
commit cf7ff9eea0eb22f8946908f5a8019f67de468e08 refs/heads/master
commit 7eca197fc66d2024047e54b1ed9e8b44361a0fc2 refs/heads/tmp-directory-add
commit 642a205f37de85005a85d427b53ee4fb2252e82e refs/heads/tmp/generic-releases
tag 20f043b1379cf768d966597799fd4907c757f755 refs/tags/v0.0.1
tag 72a21991a384e539096dbb867bfb0bee72aee2cd refs/tags/v0.0.10
tag 3590e0ca0ebb070e5b376705fa230bbfa4ffa5cc refs/tags/v0.0.11
tag 33378427a403ba569a67777b8d58f6674fbc6556 refs/tags/v0.0.12
tag 06f74652755b327cf590311c2bfa036cf3b4b35d refs/tags/v0.0.13
tag 5a6325fe86ab854b581d7442667d92a11e32f3bd refs/tags/v0.0.14
tag 586fba4e580b4f5fab05f59936764cbcb1a9c7f refs/tags/v0.0.15
tag 8cd8b885f4098bf363177742bd289f660e5be51c refs/tags/v0.0.16
tag a542444ee3f0fbed35efb202fee035c809abc7d6 refs/tags/v0.0.17
tag 228a2f1650dd12222e556559462e1e06fc4993d9 refs/tags/v0.0.18
tag 606979a4ca05d497fc0d24aad00dce82636ef47c refs/tags/v0.0.19
tag 32bf5a59fc2a323baa6d5f15a6ad5382ec275a67 refs/tags/v0.0.2
tag 3147c3d31ec46cf6492f881e908b1237ebdff2c7 refs/tags/v0.0.20
tag 215ea50daba111e082e0b72e76eb4b6073a87908 refs/tags/v0.0.21
tag 3fb168c2072a5d6252124257a1e5dfc0f5ffa1df refs/tags/v0.0.22
tag 8cdbee8da4d73fc5d262789e460a16ac3c72aba4 refs/tags/v0.0.23
...
```

id: b464cad1b66fff266a37b46ea6e7a04b545e904b

A revolutionary infrastructure *designed for software source code*

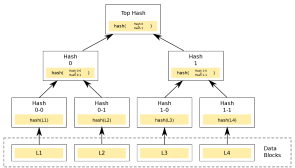
The *graph* of public software development



All software development in a single graph …

- enable traceability

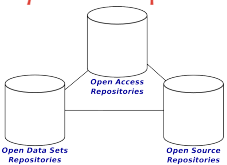
The *global ledger* of public code



… a Merkle graph

- ensure integrity

A *pillar* of Open Science

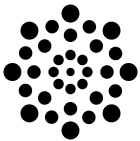


Reference archive of Research Software

- reproducibility
- reference

Reference platform for *Big Code*

uniform data structure



- large scale studies
- cybersecurity
- machine learning, AI, …

Outline

Extrinsic vs Intrinsic identifiers

In a nutshell (for more info see this dedicated blog post)

Main difference: how the *relation* between *identifier* and *designated object* is created and maintained. *Persistence* is a key desired property.

	Extrinsic	**Intrinsic**
relation	register	convention
persistence	external[1]	internal
pre-internet	passport number, ISBN, SSN, etc.	Music/Chemistry notations *e.g. NaCl is table salt*
internet era	DOI, Handle, Ark, etc.	cryptographic hashes *e.g.: git, bitcoin, SWHID*

distributed software development (e.g. git) relies on *intrinsic identifiers* we need a *uniform* intrinsic identifier

[1] "persistence... is a function of *administrative* care" RFC 3650 (Handle System Overview, 2003)

A DOI pointing to a Zenodo *zip*

- resolves to a snapshot you must *trust* — no built-in integrity check
- one opaque blob: no *history*, no *diff*, nothing addressable *inside* it
- granularity stops at "the whole deposit"

A *SWHID*

- intrinsic: the identifier *is* the cryptographic hash → self-verifying
- addresses *any* level: snapshot / revision / directory / file / lines
- works for *any* VCS (git, hg, svn) and for plain tarballs

a DOI answers *"which deposit?"*; a SWHID answers *"exactly which bytes, and prove it"*

Outline

A SWHID is a *content-derived* identifier: every implementation must agree, or it is a bug.

```
# the Python CLI you will actually type (Software Heritage, GPLv3)
$ pip install 'swh.model[cli]'
$ swh identify --type directory some/tree/
```

The reference implementation is `swhid-rs` (Rust, SWHID Working Group). Also: OCaml, Go, Ruby, a JS grammar — and a conformance test suite.

Catalogue: swhid.org/implementations *Standard:* ISO/IEC 18670:2025

The citation, as it appears in the bibliography

[SW Rel.] The Lua team, Lua version 1.0, July 28, 1993. PUC-Rio. vcs: `https://github.com/lua/lua`, swhid: ⟨`swh:1:dir:f2f23a63f14744c612f4cf2e52497352a3d8ef53`⟩

Everything that follows needs only *two* shell variables

```
# where the archive lives
$ ARCHIVE=https://archive.softwareheritage.org

# what we are checking: the Lua 1.0 source tree, exactly as cited above
$ SWHID=swh:1:dir:f2f23a63f14744c612f4cf2e52497352a3d8ef53
```

This is the whole trick: the identifier travels in the paper — and anyone can recompute it from the bytes, with no account and no permission.

The archive stores a *graph*, not tarballs. Ask it to cook one for you — then wait.

```
# ask for it: POST creates the cooking task
$ curl -X POST "$ARCHIVE/api/1/vault/flat/$SWHID/"
{"id": 1234, "status": "new",
 "swhid": "swh:1:dir:f2f23a63...",
 "fetch_url": ".../vault/flat/swh:1:dir:f2f23a63.../raw/"}

# poll until it is ready: new -> pending -> done   (or failed)
$ curl -s "$ARCHIVE/api/1/vault/flat/$SWHID/" | jq -r .status
done
```

Ask before you fetch. A GET on a directory nobody has cooked returns 404: `Cooking of swh:1:dir:... was never requested.` Add `?email=you@example.org` and the archive tells you when it is done.

```
# 1. download only once status said "done"
#    (-OJ keeps the server's filename)
$ curl -LOJ "$ARCHIVE/api/1/vault/flat/$SWHID/raw/"

# 2. unpack: one top-level dir, named after the SWHID
$ tar xf swh_1_*.tar.gz
$ DIR=$(tar tf swh_1_*.tar.gz | head -1)

# 3. recompute locally: offline, no account needed
$ swh identify --type directory "$DIR"
swh:1:dir:f2f23a63f14744c612f4cf2e52497352a3d8ef53
```

Same string as the $SWHID you started from. **You did not have to trust the archive** — or us.

Outline

Four steps, on *Lua*

1. Browse — Lua is already archived
   last full visit: 30 August 2026
2. Refresh it yourself — Save Code Now, or the updateswh browser extension
3. Get a SWHID — pick any file, any line
4. Cite it — biblatex-software, Overleaf ACMART template

Why Lua

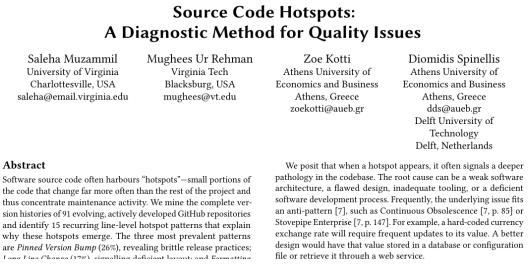
Born at PUC-Rio. First contribution 1993-07-28 — the oldest project in the Brazilian academic record. 9355 stars, still active in 2026.

[SW Rel.] The Lua team, Lua version 1.0, July 28, 1993. PUC-Rio. vcs: `https://github.com/lua/lua`, swhid: ⟨`swh:1:dir:f2f23a63f14744c612f4cf2e52497352a3d8ef53`⟩

Lua 1.0 — *"It was never publicly released."* Archived anyway, and citable forever.

SWHIDs in published research — today, not hypothetically

D. Spinellis *et al.* — MSR 2026 — 30 SWHIDs — IEEE TSE, submitted — 57 SWHIDs



Source Code Hotspots:
A Diagnostic Method for Quality Issues

Saleha Muzammil, University of Virginia, Charlottesville, USA, saleha@email.virginia.edu
Mughees Ur Rehman, Virginia Tech, Blacksburg, USA, mughees@vt.edu
Zoe Kotti, Athens University of Economics and Business, Athens, Greece, zoekotti@aueb.gr
Diomidis Spinellis, Athens University of Economics and Business, Athens, Greece, dds@aueb.gr; Delft University of Technology, Delft, Netherlands

Abstract

Software source code often harbours "hotspots"—small portions of the code that change far more often than the rest of the project and thus concentrate maintenance activity. We mine the complete version histories of 91 evolving, actively developed GitHub repositories and identify 15 recurring line-level hotspot patterns that explain why these hotspots emerge. The three most prevalent patterns are *Pinned Version Bump* (26%), revealing brittle release practices;

We posit that when a hotspot appears, it often signals a deeper pathology in the codebase. The root cause can be a weak software architecture, a flawed design, inadequate tooling, or a deficient software development process. Frequently, the underlying issue fits an anti-pattern [7], such as Continuous Obsolescence [7, p. 85] or Stovepipe Enterprise [7, p. 147]. For example, a hard-coded currency exchange rate will require frequent updates to its value. A better design would have that value stored in a database or configuration file or retrieve it through a web service.

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. X, NO. Y, AUGUST 2026 1

You Are not Expected to Understand this:
The Usage of the C Preprocessor in the Linux Kernel

Diomidis Spinellis, *Senior Member, IEEE*

***Abstract*—The Linux kernel is a foundational element of the modern IT infrastructure, powering billions of devices from smartphones and routers to desktops and supercomputers. Consequently, the quality of its code affects IT's reliability, resilience, performance, and evolution. In this exploratory case study we aim to improve the understanding of the Linux kernel's code quality regarding its usage of the C preprocessor; a dated, ubiquitous, and often insidious part of the compilation process,**

targeting the Intel x86 CPUs, nowadays it supports modules and several processor architectures. Linux is mainly written in the C programming language, with the latest version 6.14.2 we analyzed comprising 35 million lines of C code, 377 thousand lines of assembly code, and 30 thousand lines of Rust code[1].

An important though idiosyncratic programming language feature of C (and C++) is the preprocessing directives [5]

```
swh:1:cnt:97c3d084ebf023031226faf869652ee65af59337;
    anchor=swh:1:rev:70d7367daa048fda2c5c39d007e3b13a576419e9;
    path=usr/sys/ken/slp.c;
    lines=325
```

Ken Thompson's scheduler, line by line — and in my own 2020 ReScienceC replication

Software Heritage is a game changer

A *universal* archive

- all the source code — even 20-line snippets and data-mangling scripts
- and, above all, the code your code depends upon

A uniform *intrinsic* identifier SWHID

- independent of the development platform or technology
- a full guarantee of integrity

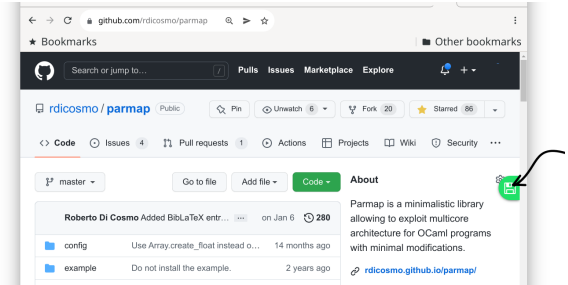
Everything else in this talk — archive, reference, describe, cite — rests on these two.

Outline

Archive in one click: the *UpdateSWH* browser extension

Check and refresh the archive from your repository page

- for Firefox, Chrome and Edge; works on GitHub, GitLab, Bitbucket, Gitea, Codeberg



What the button does

- green — up to date → open the archive / copy the SWHID
- yellow — outdated → update it (Save Code Now)
- grey — not archived yet → click to archive
- red — not archivable (private)

no account, no setup: install and click

Archive automatically: forge *webhooks*

Preserve every important version with *zero* ongoing effort

Add *one* webhook URL in your repository settings:

`https://archive.softwareheritage.org/api/1/origin/save/webhook/github/`

(also `.../gitlab/` and `.../bitbucket/`)

- recommended: trigger on *tag, branch or release creation* — not on every push
- supported: GitHub, GitLab (and instances), Bitbucket, Gitea, SourceForge

see the Software Heritage webhooks guide

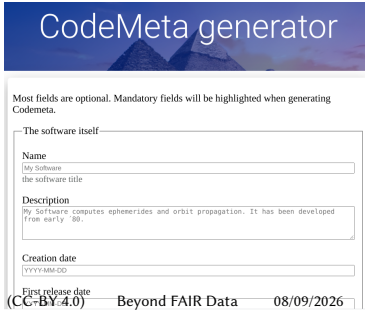
Describe your software: *CodeMeta* and CITATION.cff

Machine-readable metadata that travels *with* the code

- `codemeta.json` — a JSON-LD schema for software (authors, license, references, funding, …); Software Heritage uses it for the multi-granularity citation button — richer than CITATION.cff
- `CITATION.cff` — the lightweight cousin; GitHub renders it as a *"Cite this repository"* button — can be generated from CodeMeta

No JSON by hand: the CodeMeta generator

- fill in a web form → download a ready `codemeta.json`
  codemeta.github.io/codemeta-generator
- drop both files at the root of your repository → archived *and* indexed by Software Heritage



Cite software in your bibliography: *biblatex-software*

Four entry types, each carrying a `swhid` field, chained with `crossref` *(in TeXLive; used by ACM templates since 2022)*:

- `@software` *the project as a whole*
- `@softwaremodule` *one component*
- `@softwareversion` *a released version*
- `@codefragment` *a file, or a line range*

What the archive's *Citation* button gives you

```
@softwareversion{swh-dir-2775a5a,
  author = {Di Cosmo, Roberto and Danelutto, Marco},
  organization = {Inria and University Paris Diderot and University of Pisa},
  license = {LGPL-2.0-only}, date = {2011-07-18}, year = {2011}, month = jul,
  repository = {git+https://github.com/rdicosmo/parmap.git},
  title = {Parmap}, version = {1.2.5},
  swhid = {swh:1:dir:2775a5a092ca53c424b008fe168123c55ab1aa85;
          origin=https://github.com/rdicosmo/parmap;
          visit=swh:1:snp:6146f8cebe5245a915c21ad9186ddb0acecbd1bf;
          anchor=swh:1:rev:9022bdc093f3540cede505084969311473adbb09}
}
```

… and what *biblatex-software* renders with the `shortswhid` option

[SW Rel.] Roberto Di Cosmo and Marco Danelutto, Parmap version 1.2.5, July 18, 2011. Inria, University Paris Diderot, and University of Pisa. lic: LGPL-2.0-only. vcs: `git+https://github.com/rdicosmo/parmap.git`, swhid: ⟨swh:1:dir:2775a5a092ca53c424b008fe168123c55ab1aa85⟩

the `swhid` makes the citation *reproducible*, long after URLs break

Outline

What this means for a (PhD student/young) researcher

Starting with your next paper

- archive the exact code behind every result (Save Code Now / deposit / webhook)
- reference claims at the right granularity: a SWHID for the revision, directory, the file, the lines
- describe it: a `codemeta.json` (and a generated `CITATION.cff`) in every repository
- cite and credit it in the bibliography, like any other research output, use and promote biblatex-software

As a reviewer or AEC member

- ask authors for *archival proof* (a SWHID), not a fragile URL
- one habit, applied every time, changes the norms of your community

The floor is yours

don't treat your code like an afterthought: *record, share and value it*

the tools are ready — starting with your next paper

Explore deeper in the doctoral course notes

The Source Code of Science

References

UNESCO, *Draft recommendations on Open Science* 2021, (online)

French Ministry of Research, *Second National Plan for Open Science* 2021, (online)

EOSC SIRS Task Force, *Scholarly Infrastructures for Research Software* 2020, Publications office of the European Commission, (10.2777/28598)

R. Di Cosmo, *Archiving and Referencing Source Code with Software Heritage* International Conference on Mathematical Software 2020 (10.1007/978-3-030-52200-1_36)

J.F. Abramatic, R. Di Cosmo, S. Zacchiroli, *Building the Universal Archive of Source Code* CACM, October 2018 (10.1145/3183558)

these slides