Archiving and referencing source code with Software Heritage

Roberto Di Cosmo

Archiving and referencing source code with Software Heritage

Roberto Di Cosmo[1 [0000−0002−7493−5349]]

Software Heritage, Inria and University of Paris, France roberto@dicosmo.org

Abstract. Software, and software source code in particular, is widely used in modern research. It must be properly archived, referenced, described and cited in order to build a stable and long lasting corpus of scientific knowledge. In this article we show how the Software Heritage universal source code archive provides a means to fully address the first two concerns, by archiving seamlessly all publicly available software source code, and by providing *intrinsic persistent identifiers* that allow to reference it at various granularities in a way that is at the same time convenient and effective.
We call upon the research community to adopt widely this approach.

Keywords: Software source code · archival · reference · reproducibility

1 Introduction

Software source code is *an essential research output*, and there is a growing general awareness of its importance for supporting the research process [6, 27, 20]. Many research communities focus on the issue of *scientific reproducibility* and strongly encourage making the source code of the artefact available by archiving it in publicly-accessible long-term archives; some have even put in place mechanisms to assess research software, like the *Artefact Evaluation* process introduced in 2011 and now widely adopted by many computer science conferences [7], and the *Artifact Review and Badging* program of the ACM [4]. Other raise the complementary issues of making it easier to discover existing research software, and giving academic credit to authors [25, 21, 22].

These are important issues that are similar in spirit to those that led to the current FAIR data movement~[28], and as a first step it is important to clearly identify the different concerns that come into play when addressing software, and in particular its source code, as a research output. They can be classified as follows:

Archival: software artifacts must be properly **archived**, to ensure we can *retrieve* them at a later time;
Reference: software artifacts must be properly **referenced** to ensure we can *identify* the exact code, among many potentially archived copies, used for reproducing a specific experiment;
Description: software artifacts must be equipped with proper **metadata** to make it easy to *find* them in a catalog or through a search engine;
Citation: research software must be properly **cited** in research articles in order to give *credit* to the people that contributed to it.

As already pointed out in the literature, these are not only different concerns, but also *separate* ones. Establishing proper *credit* for contributors via *citations* or providing proper metadata to *describe* the artifacts requires a *curation* process [5, 2, 18] and is way more complex than simply providing stable, intrinsic identifiers to *reference* a precise version of a software source code for reproducibility purposes [21, 3, 16]. Also, as remarked in [20, 3], resarch software is often a thin layer on top of a large number of software dependencies that are developed and maintained outside of academia, so the usual approach based on institutional archives is not sufficient to cover all the software that is relevant for reproducibility of research.

In this article, we focus on the first two concerns, *archival* and *reference*, showing how they can be adressed fully by leveraging the Software Heritage universal archive [1], and also mention some recent evolutions in best practices for embedding *metadata* in software development repositories.

In Section 2 we briefly recall what is Software Heritage and what makes it special; in Section 3 we show how researchers can easily ensure that any relevant source code is archived; in Section 4 we explain how to use the *intrinsic identifiers* provided by Software Heritage to enrich research articles, making them more useful and appealing for the readers, and providing stable links between articles and source code in the web of scientific knowledge we are all building. Finally, we point to ongoing collaborations and future perspectives in Section 5.

2 Software Heritage: the universal archive of software source code

Software Heritage [17, 1] is a non profit initiative started by Inria in partnership with UNESCO, to build a long term universal archive specifically designed for software source code, and able to store not only a software artifact, but *also its full development history.*

Software Heritage's mission is to collect, preserve, and make easily accessible the source code of *all* publicly available software. Among the strategies designed for collecting the source code there is the development of a large scale automated crawler for source code, whose architecture is shown in Figure 1.

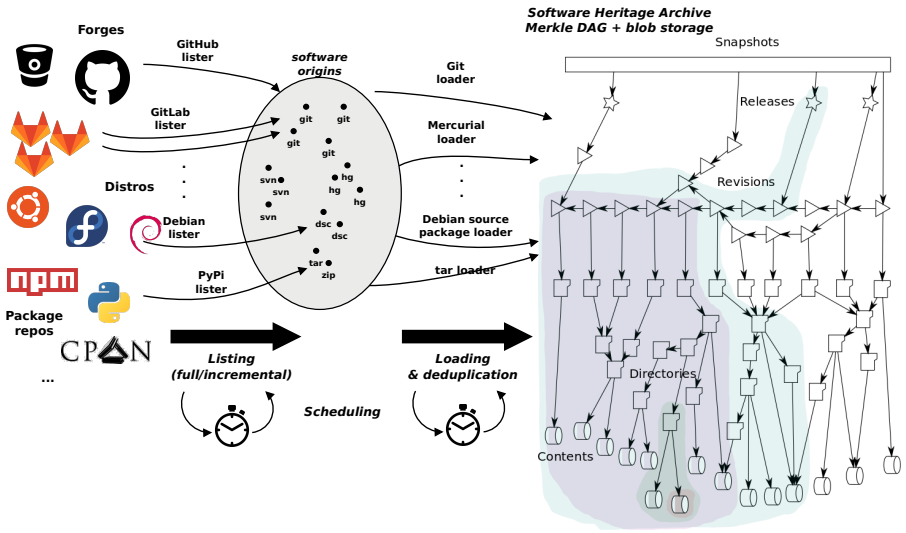


Fig. 1: Architecture of the Software Heritage crawler

The sustainability plan is based on several pillars. The first one is the support of Inria, a national research institution that is involved for the long term. A second one is the fact that Software Heritage provides a common infrastructure catering to the needs of a variety of stakeholders, ranging from industry to academia, from cultural heritage to public administrations. As a consequence, funding comes from a diverse group of sponsors, ranging from IT companies to public institutions.

Finally, an extra layer of security is provided by a network of independed, international mirrors that maintain a full copy of the archive [1].

We recall here a few key properties that set Software Heritage apart from all other scholarly infrastructures:

[1] More details can be found at `https://www.softwareheritage.org/support/sponsors` and `https://www.softwareheritage.org/mirrors`.

– it *proactively* archives *all software*, making it possible to store and reference any piece of publicly available software relevant to a research result, independently from any specific field of endeavour, and even when the author(s) did not take any step to have it archived [17, 1];

– it stores the source code with its development history in a uniform data structure, a Merkle DAG [23], that allows to provide uniform, *intrinsic* identifiers for the billions of software artifacts of the archive, independently of the version control system or package format used [16].

At the time of writing this article, the Software Heritage archive contains over 7 billions unique source code files, from more than 100 million different software origins[2].

It provides the ideal place to *preserve research software artifacts*, and offers powerful mechanisms to *enhance research articles* with precise references to relevant fragments of your source code. Using Software Heritage is straightforward and involves very simple steps, that we detail in the following sections.

3 Archiving and self archiving

In a research article one may want to reference different kinds of source code artifacts: some may be popular open source components, some may be general purpose libraries developed by others, and some may be one's own software projects.

All these different kinds of software artifacts can be archived extremely easily in Software Heritage: it's enough that their source code is hosted on a publicly accessible repository (Github, Bitbucket, any GitLab instance, an institutional software forge, etc.) using one of the version control systems supported by Software Heritage, currently Subversion, Mercurial and Git [3].

For source code developed on popular development platforms, chances are that the code one wants to reference is already archived in Software Heritage, but one can make sure that the archived version history is fully up to date, as follows:

– go to `https://save.softwareheritage.org`,
– pick the right version control system in the drop-down list, enter the code repository url [4],
– click on the Submit button (see Figure 2).

Fig. 2: The Save Code Now form

That's all. No need to create an account or disclose personal information of any kind. If the provided URL is correct, Software Heritage will archive the repository shortly after, with its full development history. If it is hosted on one of the major forges we already know, this process will take just a few hours; if it is in a location we never saw before, it can take longer, as it will need to be manually screened [5].

[2] See `https://archive.softwareheritage.org` for the up to date figures.
[3] For up to date information, see `https://archive.softwareheritage.org/browse/origin/save/`
[4] Make sure to use the clone/checkout url as given by the development platform hosting your code. It can easily be found in the web interface of the development platform.
[5] It is also possible to request archival programmatically, using the Software Heritage API, which can be quite handy to integrate in a Makefile; see `https://archive.softwareheritage.org/api/1/origin/save/` for details.

3.1 Preparing source code for self archiving

In case the source code is one own's, before requesting its archival it is important to structure the software repository following well established good practices for release management [24]. In particular one should add README and AUTHORS files as well as licence information following industry standard terminology [19, 26].

Future users that find the artifact useful might want to give credit by citing it. To this end, one might want to provide instructions on how one prefers the artifact to be cited. We would recommend to also provide structured metadata information in machine readable formats. While practices in this area are still evolving, one can use the CodeMeta generator available at `https://codemeta.github.io/codemeta-generator/` to produces metadata conformant to the CodeMeta schema: the JSON-LD output can be put at the root of the project in a **codemeta.json** file. Another option is to use the Citation File Format, CFF (usually in a file named **citation.cff**).

4 Referencing

Once the source code has been archived, the Software Heritage *intrinsic identifiers*, called SWH-ID, fully documented online and shown in Figure 3, can be used to reference with great ease any version of it.

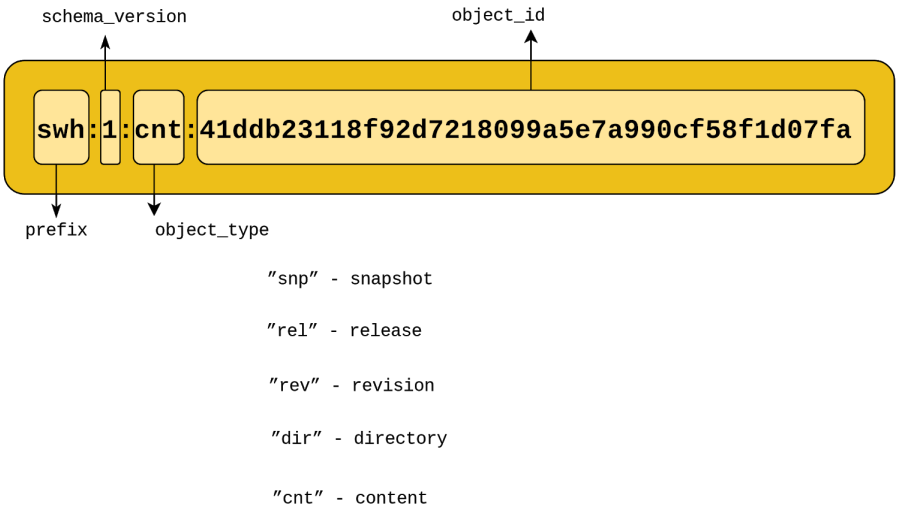


Fig. 3: Schema of the core Software Heritage identifiers

SWH-IDs are URIs with a very simple schema: the `swh` prefix makes explicit that these identifiers are related to Software Heritage; the colon (`:`) is used as separator between the logical parts of identifiers; the schema version (currently `1`) is the current version of this identifier schema; then follows the type of the objects identified and finally comes a hex-encoded (using lowercase ASCII characters) cryptographic signature of this object, computed in a standard way, as detailed in [15, 16].

These core identifiers may be equipped with the *qualifiers* that carry contextual *extrinsic* information about the object:

origin : the *software origin* where an object has been found or observed in the wild, as an URI;

visit : persistent identifier of a *snapshot* corresponding to a specific *visit* of a repository containing the designated object;

anchor : a *designated node* in the Merkle DAG relative to which a *path to the object* is specified;

path : the *absolute file path*, from the *root directory* associated to the *anchor node*, to the object;

lines : *line number(s)* of interest, usually within a content object

The combination of the core SWH-IDs with these qualifiers provides a very powerful means of referring in a research article to all the software artefacts of interest.

To make this concrete, in what follows we use as a running example the article *A "minimal disruption" skeleton experiment: seamless map and reduce embedding in OCaml* by Marco Danelutto and Roberto Di Cosmo [9] published in 2012. This article introduced `Parmap`[12], an elegant library for multicore parallel programming that was distributed via the `gitorious.org` collaborative development platform, at `gitorious.org/parmap`. Since Gitorious has been shut down a few years ago, like Google Code and CodePlex, this example is particularly fit to show why pointing to an *archive* of the code is better than pointing to the collaborative development platform where it is developed.

4.1 Specific version

The Parmap article describes a *specific version* of the Parmap library, the one that was used for the experiments reported in the article, so in order to support reproducibility of these results, we need to be able to pinpoint precisely the state(s) of the source code used in the article.

The exact revision of the source code of the library used in the article has the following SWH-ID:

```
swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;
origin=https://gitorious.org/parmap/parmap.git;
visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82
```

This identifier can be turned into a clickable URL by prepending to it the prefix `https://archive.softwareheritage.org/` (one can try it by clicking on this link).

4.2 Code fragment

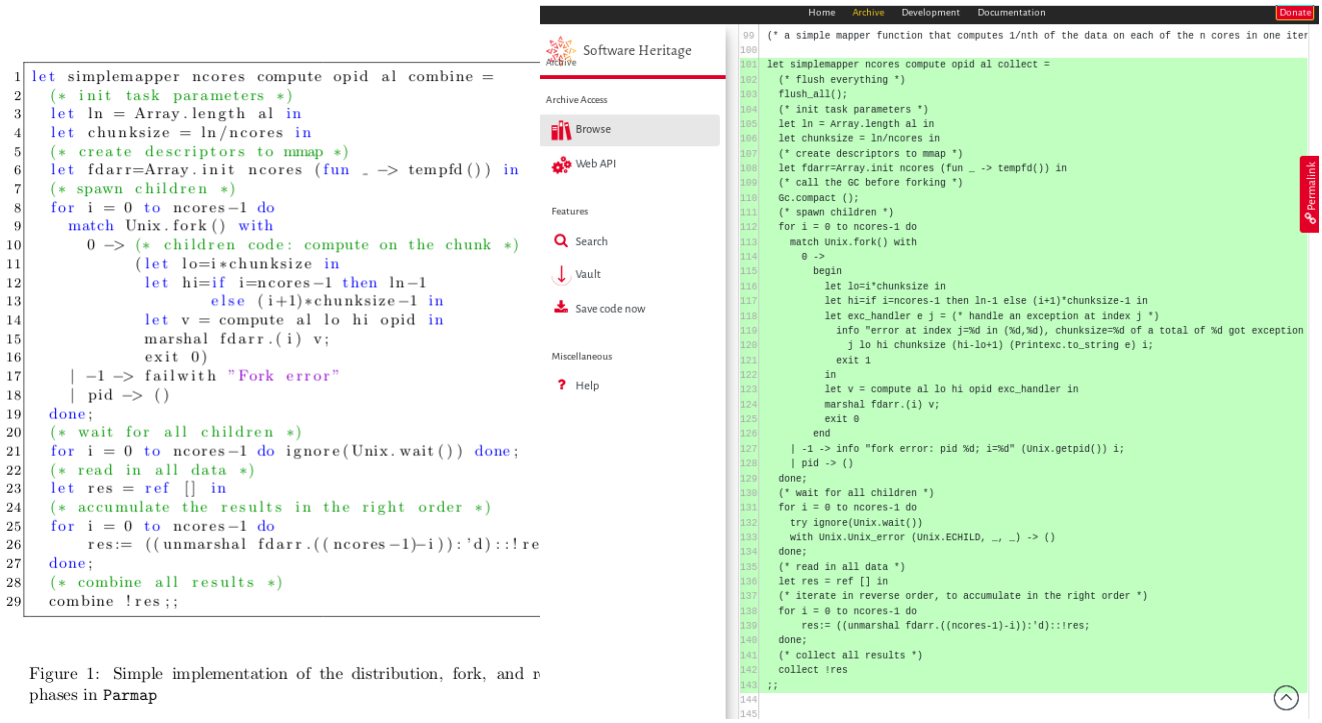
Having a link to the exact archived revision of a software project is important in all research articles that use software, and the core SWH-IDs allow to drill down and point to a given directory or even a file content, but sometimes, like in our running example, one would like to do more, and pinpoint a fragment of code inside a specific version of a file. This is possible using the `lines=` qualifier available for identifiers that point to file content.

Let's see this feature at work in our running example, showing how the experience of studying or reviewing an article can be greatly enhanced by providing pointers to code fragments.

In Figure 1 of [9], which is shown here as Figure 4a, the authors want to present the core part of the code implementing the parallel functionality that constitutes the main contribution of their article. The usual approach is to typeset in the article itself *an excerpt of the source code*, and let the reader try to find it by delving into the code repository, which may have evolved in the mean time. Finding the exact matching code can be quite difficult, as the code excerpt is *often edited* a bit with respect to the original, sometimes to drop details that are not relevant for the discussion, and sometimes due to space limitations.

In our case, the article presented 29 lines of code, slightly edited from the 43 actual lines of code in the Parmap library: looking at 4a, one can easily see that some lines have been dropped (102-103, 118-121), one line has been split (117) and several lines simplified (127, 132-133, 137-142).

Using Software Heritage, the authors can do a much better job, because the original code fragment can now be precisely identified by the following Software Heritage identifier:



```
let simplemapper ncores compute opid al combine =
  (* init task parameters *)
  let ln = Array.length al in
  let chunksize = ln/ncores in
  (* create descriptors to mmap *)
  let fdarr=Array.init ncores (fun _ -> tempfd()) in
  (* spawn children *)
  for i = 0 to ncores-1 do
    match Unix.fork() with
      0 -> (* children code: compute on the chunk *)
           (let lo=i*chunksize in
            let hi=if i=ncores-1 then ln-1
                   else (i+1)*chunksize-1 in
            let v = compute al lo hi opid in
            marshal fdarr.(i) v;
            exit 0)
    | -1 -> failwith "Fork error"
    | pid -> ()
  done;
  (* wait for all children *)
  for i = 0 to ncores-1 do ignore(Unix.wait()) done;
  (* read in all data *)
  let res = ref [] in
  (* accumulate the results in the right order *)
  for i = 0 to ncores-1 do
      res:= ((unmarshal fdarr.((ncores-1)-i)):'d)::!re
  done;
  (* combine all results *)
  combine !res;;
```

Figure 1: Simple implementation of the distribution, fork, and r
phases in Parmap

(a) as presented in the article [9]

(b) as archived in Software Heritage

Fig. 4: Code fragment from the published article compared to the content in the Software Heritage archive

swh:1:cnt:d5214ff9562a1fe78db51944506ba48c20de3379;
origin=https://gitorious.org/parmap/parmap.git;
visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82;
anchor=swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;
path=/parmap.ml;
lines=101-143

This identifier will **always** point to the code fragment shown in Figure 4b.

The caption of the original article shown in Figure 4a can then be significantly enhanced by incorporating a clickable link containing the SWH-ID shown above: it's all is needed to point to the exact source code fragment that has been edited for inclusion in the article, as shown in Figure 5. The link contains, thanks to the SWH-ID qualifiers, all the contextual information necessary to identify the context in which this code fragment is intended to be seen.

Simple implementation of the distribution, fork, and recollection phases in `Parmap` (slightly simplified from the the actual code in the version of Parmap used for this article)

Fig. 5: A caption text with the link to the code fragment and its contextual information

When clicking on the hyperlinked text in the caption shown above, the reader is brought seamlessly to the Software Heritage archive on a page showing the corresponding source code archived in Software Heritage, with the relevant lines highlighted (see Figure 4b).

4.3 Software bibliographies with `biblatex-software`

Another way to enrich an article with precise pointers to software source code is by adding entries for it in the bibliography. Unfortunately, standard bibliography styles do not treat software as a first class citizen, and for example BibTeX users often resort to the `@misc` entry to this end, which is really unsatisfactory.

Since April 2020, users of the BibLaTeX package can leverage the `biblatex-software` package [10], available on CTAN [8], to produce rich software bibliographies.

This package support four kind of different entries:

- `@software` for describing the general information about a software project
- `@softwareversion` for describing a specific version or release of a software project
- `@softwaremodule` for describing a module that is part of a larger software project
- `@codefragment` for describing a fragment of code (full file, or selected lines of a file)

Using these special BibTeX entries, the various examples presented in the previous sections above can be described as follows

```
@software {parmap,
 title = {The Parmap library},
 author = {Di Cosmo, Roberto and Marco Danelutto},
 year = {2012},
 institution = {{University Paris Diderot} and {University of Pisa}},
 url = {https://rdicosmo.github.io/parmap/},
 license = {LGPL-2.0},
}

@softwareversion {parmap-0.9.8,
 title = {The Parmap library},
 author = {Di Cosmo, Roberto and Marco Danelutto},
 version = {0.9.8},
 swhid = {swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;
  origin=https://gitorious.org/parmap/parmap.git;
  visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82},
 crossref = {parmap}
}

@codefragment {simplemapper,
 subtitle = {Core mapping routine},
 swhid = {
  swh:1:cnt:d5214ff9562a1fe78db51944506ba48c20de3379;
  origin=https://gitorious.org/parmap/parmap.git;
  visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82;
  anchor=swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;
  path=/parmap.ml;
  lines=101-143},
 crossref = {parmap-0.9.8}
}
```

The result can be seen in the bibliography of this article as [13] and [14].

4.4 Getting the SWH-ID

A fully qualified SWH-ID is rather long, and it needs to be, as it contains quite a lot of information that is essential to convey. In order to make it easy to use SWH-IDs, we provide a very simple way of getting the right SWH-ID without having to type it by hand. Just browse the archived code in Software Heritage and navigate to the software artifact of interest. Clicking on the *permalinks vertical red tab* that is present on all pages of the archive, opens up a tab that allows to select the identifier for the object of interest: an example is shown in Figure 6.

The two buttons on the botton right allow to copy the identifier or the full permalink in the clipboard, and to paste it in an article as needed.

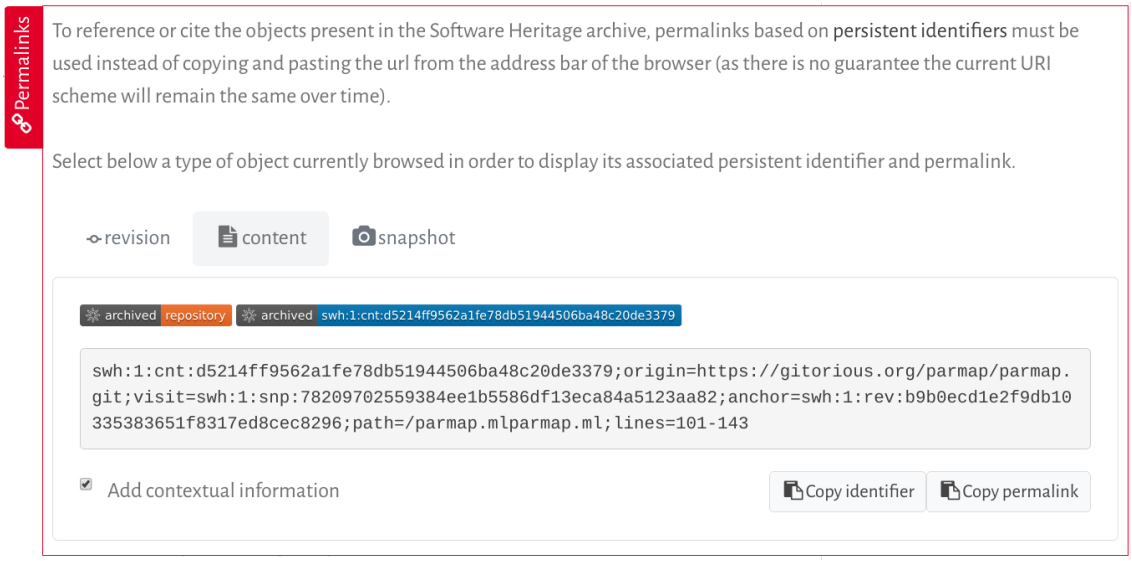


Fig. 6: Obtaining a Software Heritage identifier using the permalink box on the archive Web user interface

4.5 Generating and verifying SWH-IDs

An important consequence of the fact that SWH-IDs are *intrinsic identifiers* is that they can be generated and verified *independently* of Software Heritage, using `swh-identify`, an open source tool developed by Software Heritage, and distributed via PyPI as `swh.model`, with the stable version at the time of writing being this one.

Version 1 of the SWH-IDs uses git-compatible hashes, so if the source code that one wants to reference uses git as a version control system, one can create the right SWH-ID by just prepending `swh:1:rev:` to the commit hash. This comes handy to automate the generation of the identifiers to be included in an article, as one will always have code and article in sync.

5 Perspectives for the scholarly world

We have shown how Software Heritage and the associated SWH-IDs enables the seamless archival of all publicly available source code. It provides for all kind of software artifacts the *intrinsic identifiers* that are needed to establish long lasting, resilient links between research articles and the software they use or describe.

All researchers can use *right now* the mechanisms presented here to produce improved and enhanced research articles. More can be achieved by establishing collaborations with academic journals, registries and institutional repositories and registries, in particular in terms of description and support for software citation. Among the initial collaborations that have been already established, we are happy to mention the cross linking with the curated mathematical software descriptions maintained by the `swMath.org` portal [5], and the curated deposit of software artefacts into the HAL french national open access portal [18], which is performed via a standard SWORD protocol inteface, an approach that is currently being explored by other academic journals.

We believe that the time has come to see software become a first class citizen in the scholarly world, and Software Heritage provides a unique infrastructure to support an open, non profit, long term and resilient web of scientific knowledge.

5.1 Acknowledgements

This article is a major evolution of the research software archival and reference guidelines available on the Software Heritage website [11] resulting from extensive discussions that took place over several years with many people. Special thanks to Alain Girault, Morane Gruenpeter, Antoine Lambert, Julia Lawall, Arnaud Legrand, Nicolas Rougier and Stefano Zacchiroli for their precious feedback on these issues and/or earlier versions of this document.

References

[1] Jean-François Abramatic, Roberto Di Cosmo, and Stefano Zacchiroli. "Building the Universal Archive of Source Code". In: *Communications of the ACM* 61.10 (Sept. 2018), pp. 29–31.

[2] Alice Allen and J Schmidt. "Looking Before Leaping: Creating a Software Registry". In: *Journal of Open Research Software* 3.e15 (2015).

[3] Pierre Alliez et al. "Attributing and Referencing (Research) Software: Best Practices and Outlook From Inria". In: *Computing in Science Engineering* 22.1 (Jan. 2020). Available from `https://hal.archives-ouvertes.fr/hal-02135891`, pp. 39–52.

[4] Association for Computing Machinery. *Artifact Review and Badging*. `https://www.acm.org/publications/policies/artifact-review-badging`. Retrieved April 27th 2019. Apr. 2018.

[5] Sebastian Bönisch et al. "swMATH - A New Information Service for Mathematical Software". In: *MKM/Calculemus/DML*. Vol. 7961. Lecture Notes in Computer Science. Springer, 2013, pp. 369–373.

[6] Christine L. Borgman, Jillian C. Wallis, and Matthew S. Mayernik. "Who's Got the Data? Interdependencies in Science and Technology Collaborations". In: *Computer Supported Cooperative Work* 21.6 (2012), pp. 485–523.

[7] Bruce R. Childers et al. "Artifact Evaluation for Publications (Dagstuhl Perspectives Workshop 15452)". In: *Dagstuhl Reports* 5.11 (2016). Ed. by Bruce R. Childers et al., pp. 29–35.

[8] *CTAN: the Comprehensive TeX Archive Network*. URL: `http://www.ctan.org/` (visited on 04/29/2020).

[9] Marco Danelutto and Roberto Di Cosmo. "A "Minimal Disruption" Skeleton Experiment: Seamless Map & Reduce Embedding in OCaml". In: *Procedia CS* 9 (2012), pp. 1837–1846.

[10] [SW] Roberto Di Cosmo, *BibLaTeX stylefiles for software products*, 2020. URL: `https://ctan.org/tex-archive/macros/latex/contrib/biblatex-contrib/biblatex-software`.

[11] Roberto Di Cosmo. "How to use Software Heritage for archiving and referencing your source code: guidelines and walkthrough". Available at `https://hal.archives-ouvertes.fr/hal-02263344`. Apr. 2019.

[12] [SW] Roberto Di Cosmo and Marco Danelutto, *The Parmap library*, 2012. University Paris Diderot and University of Pisa. LIC: LGPL-2.0. URL: `https://rdicosmo.github.io/parmap/`.

[13] [SW REL.] Roberto Di Cosmo and Marco Danelutto, *The Parmap library* version 0.9.8, 2012. University Paris Diderot and University of Pisa. LIC: LGPL-2.0. SWHID: ⟨`swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;origin=https://gitorious.org/parmap/parmap.git;visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82`⟩.

[14] [SW EXC.] Roberto Di Cosmo and Marco Danelutto, "Core mapping routine", from *The Parmap library* version 0.9.8, 2012. University Paris Diderot and University of Pisa. LIC: LGPL-2.0. SWHID: ⟨`swh:1:cnt:d5214ff9562a1fe78db51944506ba48c20de3379;origin=https://gitorious.org/parmap/parmap.git;visit=swh:1:snp:78209702559384ee1b5586df13eca84a5123aa82;anchor=swh:1:rev:0064fbd0ad69de205ea6ec6999f3d3895e9442c2;path=/parmap.ml;lines=101-143`⟩.

[15] Roberto Di Cosmo, Morane Gruenpeter, and Stefano Zacchiroli. "Identifiers for Digital Objects: the Case of Software Source Code Preservation". In: *Proceedings of the 15th International Conference on Digital Preservation, iPRES 2018, Boston, USA*. Sept. 2018.

[16] Roberto Di Cosmo, Morane Gruenpeter, and Stefano Zacchiroli. "Referencing Source Code Artifacts: a Separate Concern in Software Citation". In: *Computing in Science & Engineering* 22.2 (Mar. 2020), pp. 33–43.

[17] Roberto Di Cosmo and Stefano Zacchiroli. "Software Heritage: Why and How to Preserve Software Source Code". In: *Proceedings of the 14th International Conference on Digital Preservation, iPRES 2017*. Sept. 2017.

[18] Roberto Di Cosmo et al. "Curated Archiving of Research Software Artifacts : lessons learned from the French open archive (HAL)". Presented at the International Digital Curation Conference, submitted to IJDC. Dec. 2019.

[19] Free Software Foundation Europe. *REUSE Software*. `https://reuse.software`. Accessed on 2019-09-24. Sept. 2019.

[20] Konrad Hinsen. "Software Development for Reproducible Research". In: *Computing in Science and Engineering* 15.4 (2013), pp. 60–63.

[21] James Howison and Julia Bullard. "Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature". In: *Journal of the Association for Information Science and Technology* 67.9 (2016), pp. 2137–2155. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.23538`.

[22] Anna-Lena Lamprecht et al. "Towards FAIR principles for research software". In: Preprint (2019). Preprint, pp. 1–23.

[23] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function". In: *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*. Ed. by Carl Pomerance. Vol. 293. Lecture Notes in Computer Science. Springer, 1987, pp. 369–378.

[24] Eric S Raymond. *Software Release Practice HOWTO*. `https://www.tldp.org/HOWTO/html_single/Software-Release-Practice-HOWTO/`. Accessed on 2019-06-05. Jan. 2013.

[25] Arfon M. Smith, Daniel S. Katz, and Kyle E. Niemeyer. "Software citation principles". In: *PeerJ Computer Science* 2:e86 (2016).

[26] SPDX Workgroup. *Software Package Data Exchange Licence List*. `https://spdx.org/license-list`, retrieved 30 March 2020. 2019.

[27] Victoria Stodden, Randall J. LeVeque, and Ian Mitchell. "Reproducible Research for Scientific Computing: Tools and Strategies for Changing the Culture". In: *Computing in Science and Engineering* 14.4 (2012), pp. 13–17.

[28] Mark D. Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific Data* 3.1 (Mar. 2016), p. 160018.