Connecter l'écosystème OCaml à Software Heritage via opam

Léo Andrès, Raja Boujbel, Louis Gesbert, Dario Pinto

Connecter l'écosystème OCaml à Software Heritage via opam

Léo Andrès[1,2], Raja Boujbel[1], Louis Gesbert[1], and Dario Pinto[1]

[1] OCamlPro SAS, 21 rue de Châtillon, 75014 Paris, France
leo.andres@ocamlpro.com, raja.boujbel@ocamlpro.com, louis.gesbert@ocamlpro.com, dario.pinto@ocamlpro.com
[2] Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Résumé

Software Heritage est un projet initié par Inria ayant pour but d'archiver l'ensemble des logiciels libres disponibles sur internet. Dans cet article nous présentons Software Heritage et décrivons nos travaux en lien avec l'écosystème OCaml, opam et Software Heritage. Ces travaux comprennent notamment l'ajout à Software Heritage de modules permettant l'archivage des paquets présents sur opam, le développement d'une bibliothèque OCaml permettent de travailler avec les identifiants Software Heritage, l'ajout à opam de la possibilité de récupérer sur Software Heritage des paquets qui ne sont plus disponibles et enfin la correction du dépôt opam officiel afin de retrouver les paquets déjà manquants. Aujourd'hui, 3516 paquets opam sont déjà archivés sur Software Heritage.

1 Software Heritage

Il arrive qu'un site web ne soit plus accessible. On peut alors le retrouver au moyen de la Wayback Machine, un site web fourni par l'Internet Archive et qui permet l'accès à des sauvegardes d'instantanés de pages webs effectuées à plusieurs dates. Malheureusement, archiver l'intégralité du web est une tâche ardue et il arrive qu'on ne puisse pas retrouver un contenu disparu par ce moyen.

Lorsque les logiciels libres disparaissent d'Internet, cela peut avoir de lourdes conséquences. Un exemple bien connu est celui d'un paquet d'une dizaine de lignes de code appelé leftpad retiré de npm par son auteur, cassant au passage des milliers d'autres paquets dont React. Mais cela arrive de bien d'autres manières :

— une personne supprime un dépôt de GitHub ;
— un chercheur part à la retraite et son site web institutionnel n'est pas maintenu ;
— une forge hébergeant de nombreux logiciels ferme (par exemple Bitbucket ne permettant plus de gérer des dépôts Mercurial, Inria fermant sa forge, rachat et fermeture de Gitorious en l'espace de trois semaines, fermeture de GoogleCode) ;
— une entreprise fait faillite et son instance GitLab n'est plus accessible.

Quelles conséquences à cela ? Vous ne pouvez plus jouer à ce petit jeu vidéo écrit par un étudiant de master qui vous amusait tant ; les résultats d'un article de recherche ne sont plus reproductibles ; des milliers de bibliothèques logicielles ne sont plus disponibles ; une autre entreprise ne peut plus maintenir son logiciel phare car elle dépendait de nombreux logiciels sur cette instance GitLab.

Il existe une solution à cela : Software Heritage. Dans la suite de l'article, vous sera présenté Software Heritage et le travail effectué par OCamlPro en lien avec l'écosystème OCaml, opam

et Software Heritage. Ce travail a été rendu possible grâce à une bourse de l'Alfred P. Sloan Foundation. Software Heritage [CZ17] est une initiative à but non lucratif, dont la mission est, selon leurs propres termes, la suivante :

> We are building the universal software archive. We collect and preserve software in source code form, because software embodies our technical and scientific knowledge and humanity cannot afford the risk of losing it. Software is a precious part of our cultural heritage. We curate and make accessible all the software we collect, because only by sharing it we can guarantee its preservation in the very long term.

À l'origine du projet, il y a Inria. Le projet Software Heritage est dirigé par Roberto Di Cosmo et Stefano Zacchiroli, deux personnes réputées pour leur implication dans le monde du logiciel libre, d'OCaml ou encore du projet Debian.

Aujourd'hui, Software Heritage gère, entre autres, l'archivage de Bitbucket, cgit, CRAN, Debian, Gitea, GitHub, GitLab, Guix, GNU, Heptapod, Launchpad, NixOS, npm, PyPI et SourceForge. Au moment de l'écriture de cet article, sont archivés environ :

- 11 milliards de fichiers sources,
- 9 milliards de dossiers,
- 2 milliards de commits,
- 165 millions de projets,
- 19 millions de versions publiées de logiciels [1].

Archiver autant de logiciels demande des ressources importantes. Software Heritage dispose de nombreux parrains lui permettant de mener à bien sa mission et garantissant sa pérénnité. On peut notamment citer Huawei, Microsoft, Intel, le CNRS, la Société Générale, le MESRI, Open Invention Network, Sorbonne Université, Université de Paris, Adacore, CAST Software, DINSIC, GitHub, Google, Università di Pisa, VMware, Dans, FOSSID, University of Bologna, Nokia Bell Labs et UQAM.

2 Gestion d'opam dans Software Heritage

Une partie du logiciel libre que l'on souhaite archiver dans Software Heritage est écrit dans le langage OCaml. À l'instar de nombreux langages, OCaml dispose d'un gestionnaire de paquets, Opam, qui permet d'installer facilement des bibliothèques et des exécutables à partir de leurs sources. Opam étant le gestionnaire de paquets de référence dans la communauté OCaml, il constitue de fait un point d'entrée idéal pour récupérer des sources OCaml. Nous présentons dans cette section un travail qui automatise l'archivage des paquets opam par Software Heritage.

Depuis quelques jours, tout le dépôt opam officiel est archivé ! Vous pouvez vérifier que votre paquet opam favori est bien archivé. Il est prévu d'archiver également des dépôts alternatifs. Ils peuvent être signalés sur discuss.ocaml.org. Il est notamment prévu que soit archivé le dépôt opam Inria hébergeant des paquets Coq. En effet, opam étant agnostique du langage utilisé par les paquets, notre travail permet d'héberger n'importe quel paquet pour n'importe quel langage, pourvu qu'un fichier opam ait été écrit.

Pour ajouter une nouvelle source de logiciels (*software origin*) à Software Heritage, il est nécessaire d'écrire deux éléments : un *lister* et un *loader*. C'est ce qui a été fait pour opam. Software Heritage étant écrit en Python, ces deux éléments doivent prendre la forme d'un

1. C'est en réalité plus mais pour des raisons techniques beaucoup de versions ne sont actuellement considérées que comme des *commits*.

module Python. Pour faciliter leur implémentation et leur maintenance, nous avons fait le choix de la simplicité : le paquet Debian `opam` a été ajouté comme dépendance à Software Heritage - dont toute l'architecture tourne sur des serveurs Debian - et nous nous servons du binaire `opam` dès que possible.

Le *lister* a pour but de lister tous les paquets disponibles. Pour chaque paquet, notre module produit une URL de la forme :

```
url = f"opam+{self.url}/packages/{page}"
```

où `self.url` est l'adresse du dépôt opam archivé et où `page` est le nom du paquet à archiver. Un simple appel à `opam list --all` est suffisant pour générer cette liste. Le code complet est disponible sur le dépôt swh-lister.

Le *loader* doit récupérer, à partir d'une URL générée par le *lister*, l'ensemble des versions disponibles d'un paquet et associer à chacune d'elles un ensemble de métadonnées et une URL vers l'archive contenant le code source. Ces informations sont récupérées en faisant des appels à `opam show` en précisant à chaque fois le champ que l'on souhaite récupérer (`url.src`, `authors`...). L'archive sera ensuite insérée automatiquement par un autre module de Software Heritage dans le graphe orienté acyclique de Merkle [PSZ19] servant à stocker tous les objets archivés. Le code complet est disponible dans le dépôt swh-loader-core.

Une fois ces deux modules implémentés et un dépôt opam choisi, Software Heritage va automatiquement et régulièrement récupérer les paquets présents sur ce dépôt. Au moment de tester le déploiement, beaucoup d'erreurs ont été rencontrées. La plupart venaient de Software Heritage, par exemple le format `.tbz` n'était pas géré. Mais beaucoup venaient du fait que des archives de paquets opam n'existaient plus, menant à une erreur 404. La plupart étaient en fait toujours disponibles dans le cache d'opam, certaines avaient été déplacées, etc. Quoi qu'il en soit, environ 80 demandes de hissage (*pull requests*) ont été ouvertes sur le dépôt opam pour corriger ces URLs.

3 La bibliothèque *swhid*

Pour permettre aux développeurs OCaml d'accéder à Software Heritage dans leurs projets, une bibliothèque OCaml *swhid* a été écrite, permettant de manipuler les identifiants persistants (*persistent identifiers*) de Software Heritage. Un *identifiant persistant* est un identifiant vers un objet archivé dans Software Heritage. En voilà un exemple :

```
swh:1:dir:4431f2b743ef6dffc837e710bdd7e2169f0c5fc9
```

Le préfixe `swh` indique tout simplement qu'il s'agit d'un *identifiant persistant* de Software Heritage. Le `1` est le numéro de version du schéma des *identifiants persistants* utilisé. Le mot-clé `dir` indique le type d'objet représenté par l'identifiant et la chaîne finale de 40 caractères est un hachage de l'objet. L'identifiant peut contenir des informations optionnelles appelées qualificatifs (*qualifiers*), par exemple :

```
swh:1:dir:4431f2b743ef6dffc837e710bdd7e2169f0c5fc9
;origin=deb://Debian/packages/freedink
;visit=swh:1:snp:f6df60118578aa9ee91672dc3958908a2bf61fdb
;anchor=swh:1:rev:db21f0afdb54c16b265754ca599869fda0ca4bfc
```

Elles ne sont pas utiles pas dans ce cadre, une explication détaillée est disponible sur la documentation des qualificatifs.

Software Heritage peut archiver cinq types d'objets différents. Voici leur description traduite depuis la documentation de Software Heritage :

- `cnt` pour `content` (aussi appelés `blobs`) : le contenu brut d'un fichier source sous forme d'une séquence d'octets, sans nom de fichier ou autre métadonnée. Le contenu des fichiers est souvent récurrent, par exemple d'une version d'un logiciel à la suivante, ou dans différents répertoires d'un même projet ou même dans des projets complètement disjoints.
- `dir` pour `directory` : une liste d'entrées nommées d'un même répertoire, chacune pointant vers d'autres artéfacts qui sont généralement des objets `cnt` ou bien des sous-répertoires. Ces entrées sont souvent associées à des métadonnées comprenant un nom et des bits de permission.
- `rev` pour `revision` (aussi appelés `commits`) : le développement logiciel au sein d'un projet spécifique est essentiellement une série de copies à partir du répertoire racine, indexées dans le temps, et qui contient la totalité du code source du projet. Le logiciel évolue quand un développeur modifie le contenu d'un ou plusieurs fichiers dans ce répertoire et enregistre ses modifications. Chaque copie enregistrée de la racine s'appelle une révision. Elle pointe vers un répertoire complètement déterminé et est équipée d'un lot arbitraire de métadonnées. Certaines sont ajoutées manuellement par le développeur (les messages de commit), d'autres sont générées par l'outil de gestion de versions (date de modification, commits précédents, etc.).
- `rel` pour `release` (aussi appelés `tags`) : toutes les révisions ne sont pas égales entres elles, et certaines sont choisies par les développeurs comme étant des jalons aussi appelées publications. Chaque *release* pointe vers le commit le plus récent dans l'historique du projet qui correspond à la-dite sortie et peut comporter des métadonnées arbitraires : nom et numéro de version, message d'annonce de sortie, des signatures cryptographiques, etc.
- `snp` pour `snapshot` : quelle que soit l'origine d'un logiciel, celle-ci offrira de multiples pointeurs vers les versions courantes du développement d'un projet. Dans le cas des systèmes de gestion de version, cela se manifeste au travers de branches (master, development, branches dites «de fonctionnalités» font évoluer un logiciel dans une direction précise) ; pour les gestionnaires de paquets aussi ce fait est visible, notamment dans l'existence de *versions (suites)* qui témoignent de la maturité individuelle des paquets qu'ils distribuent (Debian Stable, Debian Testing, ...). Le snapshot de l'origine d'un logiciel donné enregistre toutes les entrées qui s'y trouvent et ce vers quoi ces entrées pointaient à un moment donné. Par exemple, un objet snapshot peut tout aussi bien suivre le commit vers lequel la branche master pointait à un moment donné, que la sortie la plus récente d'un paquet donné dans la version stable d'une distribution.

Deux autres types d'objets spéciaux sont disponibles mais ils ne sont pas pertinents ici : `origins` et `visits`. Ils ne sont donc pas gérés dans la bibliothèque. Une explication détaillée est donnée dans la documentation des artéfacts logiciels (*software artifacts*).

3.1 Analyser syntaxiquement, valider et afficher des SWHIDs

Notre bibliothèque est capable d'analyser syntaxiquement, de valider et d'afficher des *identifiants persistants* :

```
let id = "swh:1:cnt:80131a360f0ae3d4d643f9e222591db8d4aa744c"

let () =
  match Swhid.Parse.from_string id with
  | Error e -> Format.eprintf "error: %s@." e
  | Ok id -> Format.printf "the id is: %a@." Swhid.Pp.identifier id
```

S'il y en a, les `qualificatifs` sont analysés syntaxiquement mais pas validés.

3.2 Calculer des SWHIDs

Notre bibliothèque est capable de calculer des SWHIDs pour les cinq types d'objets présentés précédemment :

```
(* some file for which we'd like to compute a SWHID *)
let content =
{|(executable
(name hello)
(modules hello))
|}

let swhid = Swhid.Compute.content_identifier content

let () =
  match swhid with
  | None -> Format.eprintf "invalid ID :S@."
  | Some swhid -> Format.printf "ID is: `%a`.@." Swhid.Pp.identifier swhid
```

À l'exécution on obtient :

```
ID is: `swh:1:cnt:f5a5bc805b67f7510d2e2eb07500f47ced8af8ca`.
```

On peut maintenant communiquer cet identifiant en sachant que le jour où notre fichier sera archivé, il aura cet identifiant. Les autres types d'objet sont plus compliqués à calculer. On laisse le lecteur curieux se référer à la documentation du module Swhid.Compute.

3.3 Télécharger des SWHIDs

Pour télécharger un objet à partir de son identifiant, une IPA (Interface de Programmation Applicative) est disponible sur Software Heritage. Il n'est généralement pas possible de récupérer directement un objet en utilisant cette API. Il faut effectuer une requête afin d'obtenir une URL où, après un petit temps de préparation, on pourra récupérer l'objet voulu. Cette procédure est assez simple pour `cnt` ou `dir` par exemple, mais plus compliquéé pour `release`, une *release* pouvant pointer vers différents types d'objets. Il faut effectuer une première requête pour obtenir l'objet vers lequel elle pointe et recommencer récursivement. De même, un `snapshot` étant un ensemble d'objets, on obtiendra un ensemble d'URLs où télécharger ces objets. Notre bibliothèque contient un module `Download` contenant une fonction pour chaque type d'objet :

```ocaml
(* an identifier we want to download (a file from FreeDink) *)
let id = "swh:1:cnt:80131a360f0ae3d4d643f9e222591db8d4aa744c"

let url =
  (* we parse the string to get a Swhid.Lang.identifier *)
  match Swhid.Parse.from_string id with
  | Error _e as e -> e
  | Ok id -> (
    (* we ask SWH for an URL from which the object can be downloaded *)
    Swhid.Download.content id

let () =
  match url with
  | Error e ->
    (* we didn't get an URL *)
    Format.eprintf
      "Can't get a download URL: %s@." e;
    exit 1
  | Ok url ->
    (* we got a valid URL ! :D *)
    Format.printf "The file can be downloaded at url `%s`.@." url
```

Et l'on obtient bien :

```
The file can be downloaded at url `https://archive.softwareheritage.org/api/1/con
↪  tent/sha1_git:80131a360f0ae3d4d643f9e222591db8d4aa744c/raw/`.
```

Une fonction `any` est aussi fournie, qui permet de récupérer une liste d'URLs pour un identifiant quelconque. La liste ne contiendra qu'un élément dans tous les cas, sauf pour `snapshot` où elle pourra en contenir plusieurs. Plus d'informations sont disponibles dans la documentation du module Swhid.Download.

4 Gestion dans opam

La dernière partie de ce travail, qui est toujours en cours, consiste à ajouter à opam la capacité de récupérer automatiquement des archives depuis Software Heritage, dans le cas où l'archive d'un paquet aurait disparu et ne serait pas non plus disponible dans le cache d'opam. Les étapes identifiées pour y parvenir sont les suivantes :

— gestion dans opam d'un champ `swhid` optionnel pour les fichiers `opam` ;

— téléchargement depuis Software Heritage de l'archive correspondant à l'identifiant contenu dans ce champ en cas de nécessité ;

— modification du dépôt opam officiel pour ajouter les SWHIDs de toutes les versions de chaque paquet.

Plusieurs solutions ont été envisagées pour la gestion du champ *swhid* dans les fichiers *opam*. L'approche initiale consistait à ajouter un nouveau type de somme de contrôle, mais cela impactait trop de parties du code. Une solution plus simple a ensuite été envisagée : ajouter

un champ *swhid* au sein du champ *url*. Elle fonctionnait correctement mais posait un problème (présent aussi dans la solution initiale) : cela n'est pas rétro-compatible et aurait empêché la mise à jour du dépôt opam officiel avec les SWHIDs avant que la majorité des clients n'aient été mis à jour. La solution retenue consiste à ajouter plutôt une *fausse* adresse au champ *mirrors* :

```
url {
  src: "..."
  mirrors: [ ... "https://swhid.opam.ocaml.org/swh:1:rev:15e2f26f2ae0f0197ce95b997b4fe024c3491f9e" ]
}
```

Les anciens clients échoueraient au moment de télécharger l'objet mais ils pourraient toujours utiliser le fichier opam. Les nouveaux clients pourraient quant à eux récupérer le SWHID et récupérer l'archive. De plus, il est prévu de gérer dès maintenant une adresse de la forme `swh:...`[2] afin de pouvoir remplacer les fausses adresses dans le futur. Ainsi, nous utiliserons directement le SWHID qui est une URL valide et dont le schéma est déjà enregistré à l'IANA, la société chargée notamment de gérer la zone racine des noms de domaines.

On ne devrait avoir à gérer que des objets de type *rel*, cependant du fait d'un bogue connu de Software Heritage, certaines versions ne sont considérées que comme des objets de type *rev*. La gestion des deux types d'objet est donc prévu.

Les parties de téléchargement et de mise à jour du dépôt opam sont relativement faciles. Le téléchargement étant déjà implémenté dans la bibliothèque *swhid*, il ne restera plus qu'à l'adapter dans opam. La mise à jour du dépôt opam sera quant à elle générée : on télécharge l'ensemble des archives, on calcule leur identifiant et on modifie le fichier opam à l'endroit idoine.

5 Conclusion

Les logiciels libres sont des objets de valeur. Dans cet article, nous avons décrit la façon dont Software Heritage les préserve et plus spécifiquement comment tous les logiciels publiés sur opam soient eux aussi correctement archivés. Des outils OCaml ont été développés, permettant d'interagir avec l'infrastructure Software Heritage au moyen d'une bibliothèque et en ajoutant à opam un mécanisme pour récupérer sur Software Heritage des logiciels disparus. Grâce à ce travail, c'est déjà plus de 3500 paquets qui ont été archivés et les auteurs de paquets opam ont maintenant la garantie que leur travail restera accessible à jamais via opam et Software Heritage. De même, les développeurs de logiciels OCaml savent désormais que les paquets dont ils dépendent ne risquent plus de disparaître.

Remerciements. Merci à Nicolas Dandrimont, Antoine R. Dumont, Antoine Lambert et Valentin Lorentz pour leur aide sur Software Heritage ; ainsi qu'à Roberto Di Cosmo et Jean-Christophe Filliâtre pour leur relecture.

2. C'est-à-dire `swh:1:...` et non pas `swh://...`.

Références

[CZ17] Roberto Di Cosmo et Stefano Zacchiroli. « Software Heritage : Why and How to Preserve Software Source Code ». In : *iPRES 2017 : 14th International Conference on Digital Preservation*. Kyoto, Japan, 25 sept. 2017. URL : https://hal.archives-ouvertes.fr/hal-01590958. published (cf. p. 2).

[PSZ19] Antoine Pietri, Diomidis Spinellis et Stefano Zacchiroli. « The Software Heritage Graph Dataset : Public software development under one roof ». In : *Proceedings of the 16th International Conference on Mining Software Repositories*. MSR '19. IEEE Press, 27 mai 2019, p. 138-142. DOI : 10.1109/MSR.2019.00030. URL : https://upsilon.cc/~zack/research/publications/msr-2019-swh.pdf. published (cf. p. 3).