Préserver et rendre identifiables les logiciels de recherche avec Software Heritage

Frédéric Santos, Baptiste Mélès, Sabrina Granger

Programming Historian en français

Préserver et rendre identifiables les logiciels de recherche avec Software Heritage (/fr/lecons/preserver-logiciels-recherche)

Sabrina Granger (https://orcid.org/0000-0002-9081-3851) , Baptiste Mélès (https://orcid.org/0000-0002-5692-083X) , et Frédéric Santos (https://orcid.org/0000-0003-1445-3871)

Cette leçon présente des modalités d'archivage adaptées aux spécificités du logiciel et à différents profils d'utilisateur·ices. Les bonnes pratiques pour citer les logiciels, ainsi que des méthodes pour explorer efficacement les fonds de Software Heritage, seront abordées.

Évaluée par les pairs (https://github.com/programminghistorian/ph-submissions/issues/616)

suivi éditorial par

- Daphné Mathelier (https://orcid.org/0000-0003-4837-0564)

évaluation par

- Chiara Mainardi
- Magaye Sall
- Martin Amouzou

parution

| 2024-11-15
modification

| 2024-12-17
difficulté

| Facile

https://doi.org/10.46430/phfr0034 (https://apis.ebsco.com/public/linkout/v1/ftf?ref=2290bb27-6287-4def-a7e7-169a61e65b87&id=229748)

Contenus

L'archivage des logiciels : une des conditions de la reproductibilité scientifique

Une recherche reproductible est une recherche plus robuste et plus transparente. Or, il existe plusieurs formes de reproductibilité : dans certaines disciplines, ce qui compte est davantage de pouvoir reproduire la méthode, que d'en obtenir les mêmes résultats. Dans cette leçon, l'accent est mis sur la reproductibilité computationnelle : c'est-à-dire la capacité à retrouver les mêmes résultats en utilisant les mêmes méthodes et les mêmes outils que l'auteur·e initial·e.

Si l'archivage et l'ouverture des données de recherche (/fr/lecons/preserver-ses-donnees-de-recherche) sont de plus en plus répandus au sein de nombreuses communautés académiques, ces pratiques ne répondent que partiellement à la question de la reproductibilité des travaux scientifiques. En effet, la capacité d'accéder aux données, et même à un protocole détaillé de traitement de ces données, ne permet pas à elle seule d'auditer les résultats d'une étude. Ainsi que le soulignent Davenport et ses co-auteur·es, comment reproduire des résultats sans accéder également au logiciel qui a permis de traiter les données ?

> « Un thème commun est l'importance croissante accordée à la « reproductibilité », qui fait l'objet de discussions dans de nombreuses disciplines [...]. Cela dépasse la question des « données », et nécessite que les logiciels et les pipelines d'analyse soient publiés dans un état utilisable conjointement aux articles.[1] »

Archiver les logiciels est donc un prérequis pour garantir un accès partagé aux auteur·es, contributeur·ices, utilisateur·ices et lecteur·ices de logiciels.[2]

Quoique essentiel à la compréhension des résultats, le logiciel n'est pourtant pas toujours identifié comme une ressource à préserver. En l'absence de réelle méthode d'archivage, les liens pointent bien souvent vers des sources déplacées ou perdues. Un article de D. Spinellis[3] illustre la courte durée de vie des URL dans les articles publiés (elle est en moyenne supérieure à celle d'un hamster (2 ans), mais inférieure à celle d'un pingouin (15 à 20 ans)). Le site web d'un individu ou même d'un laboratoire n'offre aucune garantie de pérennité, car un simple changement de fournisseur d'accès à Internet, un départ à la retraite, ou le changement de nom d'une institution, peuvent suffire à rendre tous les liens obsolètes.

Même les « forges[4] », plateformes de développement collaboratif en ligne telles que GitHub et GitLab — dont l'usage s'est très fortement accru ces dernières années, y compris dans le monde académique[5] — n'offrent pas de garantie d'accès pérenne. Leurs conditions d'utilisation peuvent être modifiées et les espaces de travail fonctionnant avec une technologie donnée ne seraient alors plus maintenus. Pire encore, ces plateformes peuvent être intégralement désactivées, coupant même aux auteur·es de logiciels tout accès à leur contenu — mésaventure qu'ont connue les utilisateur·ices de Google Code (https://perma.cc/N2HS-UVT2). Même les forges institutionnelles offrant de plus grandes garanties en termes de services peuvent être désactivées. Les forges sont donc de puissants outils de collaboration mais ne constituent pas la solution optimale pour mettre du code informatique à disposition du lectorat d'un article académique. Elles facilitent l'écriture du code, non sa préservation.

L'absence de vraies solutions d'archivage, tant pour les développeur·euses que pour les utilisateur·ices de logiciels, a donc des conséquences en chaine sur tout le système de citation académique et la reproductibilité des travaux — autrement dit, donc, sur la qualité de la science. Aussi est-il crucial de disposer d'un archivage dans un entrepôt dédié.

L'accès au code source des logiciels

Sous quelle forme les logiciels doivent-ils être archivés ? Leur conception traverse en effet trois phases : le code source, la compilation, et l'exécutable. Définissons d'abord ces termes.

Définitions : code source, exécutable, compilation

Le développement d'un logiciel commence par l'écriture d'un « code source », c'est-à-dire d'une série d'instructions écrites dans un ou plusieurs langages informatiques (par exemple Python (https://www.python.org/), R (https://www.r-project.org/), etc.), lisibles et compréhensibles par des humains. Il s'agit donc d'une représentation textuelle de l'architecture du programme et des algorithmes qui en sous-tendent le fonctionnement.

À l'autre bout de la chaine, on trouve un « exécutable » : c'est la version transformée du code source, lisible par la machine. Cette traduction pour la machine – non lisible par les humains – prend la forme d'une série de 0 et de 1 (d'où le nom de « binaire » parfois donné comme synonyme d'« exécutable »), codant les impulsions électriques nécessaires à l'exécution du programme.

La « compilation » est le processus transformant le code source écrit par des humains en un programme exécutable pour la machine. On peut donc imaginer le code source comme une recette de cuisine, l'exécutable comme le plat obtenu en suivant la recette, et la compilation comme le travail culinaire.[6]

L'archive Software Heritage, qui sera présentée ultérieurement, ne collecte que le code source, car c'est le seul composant du logiciel qui reste exploitable sur le long terme. En effet, la compilation n'est pas un processus facilement réversible : si l'on peut immédiatement passer du code source à l'exécutable, l'inverse est beaucoup plus délicat et incertain — de même qu'il ne suffit généralement pas de goûter un plat pour connaitre sa recette. De plus, le code source d'un logiciel, comme une recette de cuisine, peut être inspecté, compris, éventuellement modifié et à nouveau transmis par les personnes qui l'utilisent.

Autrement dit, renvoyer vers le code source constitue la manière la plus sûre et efficace de partager un logiciel sur le long terme.

Nous pouvons d'ailleurs illustrer le fait que le code source nous donne bien plus d'informations que le code exécutable, en observant par exemple un fragment de code de la mission Apollo (https://archive.softwareheritage.org/swh:1:cnt:0c1741c1fb0150f111625d02277407f628c31bac;origin=https://github.com/virtualagc/virtualagc;visit=swh:1:snp:cdcd2bc43331a436e8c659ba93175ef7d7eb339b;anchor=swh:1:rev:4e5d304eb7Luminary116/THE_LUNAR_LANDING.agc;lines=250-254). Parallèlement au code « efficace », voué à dicter le comportement de la machine, on voit en fin de ligne (derrière les croisillons `#`) des informations destinées au lectorat humain. Ces commentaires fournissent la « raison d'être » du code destiné à la machine : les auteurs y expliquent pourquoi ils ont jugé pertinent d'ajouter telle ou telle ligne de code. Cela permet de s'orienter rapidement dans le code, d'identifier les lignes responsables d'un certain comportement et éventuellement de les modifier.

À la compilation, les commentaires sont purement et simplement supprimés. De ce fait, si un humain essayait de lire le code exécutable, il n'aurait pas accès à ces précieuses informations.

Software Heritage, une infrastructure spécialisée dans l'archivage du code source

Software Heritage (https://www.softwareheritage.org/?lang=fr), infrastructure portée par l'Inria et l'Unesco, est dédiée à l'archivage du « patrimoine logiciel mondial de l'humanité (https://perma.cc/4RTF-KP2A) ». Les trois missions de Software Heritage sont la collecte, la préservation et le partage du code des logiciels rendus publics. Pour les raisons exposées dans la section précédente, Software Heritage conserve le code source des logiciels, c'est-à-dire la partie du logiciel compréhensible par l'être humain.[7]

Le premier avantage de Software Heritage est de fournir un point d'entrée central vers des millions de logiciels, développés et disséminés dans une multitude de plateformes. Le second avantage pour l'utilisateur est d'accéder à des collections qui sont régulièrement et automatiquement mises à jour. Ainsi, quelqu'un voulant renvoyer vers une certaine version d'un logiciel a le choix parmi ses différentes versions archivées. De plus, quelqu'un qui a développé un logiciel peut compter sur une version archivée de son travail.

Les options d'archivage

Software Heritage fournit plusieurs options pour l'archivage des logiciels :

1. Un archivage automatisé : les contenus publics des forges les plus couramment utilisées sont archivés automatiquement, à intervalles réguliers. La liste des forges dont le contenu est régulièrement « moissonné » (https://archive.softwareheritage.org/) inclut toutes les

forges les plus populaires. La plupart des logiciels « open source » (ouverts) que vous utilisez sont donc très probablement déjà archivés dans Software Heritage.

2. Un archivage manuel : il est aussi possible d'archiver un logiciel, ou de mettre à jour son archivage, manuellement. Ce service permet de mettre à jour des contenus entre deux vagues d'archivage automatisé. Il n'est pas nécessaire de créer un compte, ni d'être l'auteur·ice du logiciel pour y recourir. En revanche, si le logiciel n'est pas public, son archivage n'est pas autorisé. Cette documentation (https://docs.softwareheritage.org/#landing-preserve) détaille toutes les différentes options.

La pérennité de l'accès aux logiciels archivés est garantie par des identifiants spécialisés, les *SoftWare Hash Identifiers* (SWHID), que nous découvrirons plus dans les sections suivantes. À la différence d'un identifiant généraliste tel que le DOI (Digital Object Identifier (https://www.doi.org/)), le SWHID est dédié au logiciel. Ainsi, le SWHID permet non seulement de renvoyer vers une version donnée, mais aussi vers différents composants du logiciel, plutôt qu'au programme dans son ensemble. Un SWHID s'obtient instantanément et gratuitement (https://annex.softwareheritage.org/public/tutorials/getswhid_dir.gif) dès lors que le logiciel est archivé dans Software Heritage.

Le contenu de l'archive de Software Heritage

Software Heritage permet un accès unifié à des contenus issus d'une grande diversité de sources. Outre les plateformes de développement et de distribution les plus communément utilisées, Software Heritage fournit aussi :

- Des contenus publics de forges désactivées (https://archive.softwareheritage.org/browse/search/?q=googlecode.com&visit_type=git&with_content=true&with_visit=true) comme Google Code.
- Des contenus des forges institutionnelles basées sur GitLab (par exemple l'instance GitLab de l'IN2P3 du CNRS (https://gitlab.in2p3.fr/)) ayant fait la demande d'être inclues dans l'archive.
- Des codes sources associés à des articles, grâce aux partenariats entre Software Heritage et plusieurs revues en sciences et technologies
- Des codes sources de logiciels déposés dans l'archive ouverte nationale HAL (https://archive.softwareheritage.org/browse/search/?q=hal.archives-ouvertes.fr&visit_type=deposit&with_content=true&with_visit=true). Le dépôt dans HAL (https://hal.science/hal-01872189) permet de créer une description citable du logiciel, alors que l'archivage dans Software Heritage est principalement dédié à rendre identifiable des composants techniques du logiciel (voir le chapitre sur les identifiants pérennes). Multidisciplinaire, HAL permet de partager en libre accès les résultats de recherche, publiés ou non.

La politique documentaire de Software Heritage s'applique à l'échelle d'un gisement d'informations (forge, plateforme) et non pas logiciel par logiciel. L'enjeu est d'offrir la couverture la plus large possible. Cet objectif est motivé par la nature composite du logiciel : un logiciel de recherche peut en effet s'appuyer sur des éléments issus de communautés de l'administration publique ou de l'industrie. C'est pourquoi les archives de Software Heritage dépassent le périmètre académique.

En outre, la plus-value de Software Heritage est de préserver l'historique de développement des logiciels issus des forges, en plus de leur code source. L'historique de développement documente la création d'un logiciel et peut fournir des explications sur son fonctionnement. Voici l'exemple d'une révision (« commit ») de 2008 (https://archive.softwareheritage.org/browse/revision/64ac24e738823161693bf791f87adc802cf529ff/?origin_url=git://

git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git&snapshot=fc7706e4c177714475a4886831486ad0979983ea#swh-revision-changes-list), rédigée par Matthew Wilcox :

> « Implémentation générique des sémaphores
> Il n'y a plus d'enjeu critique de performance concernant les sémaphores ; une implémentation générique en C est donc préférable pour la maintenabilité, la débogage et l'extensibilité. Merci à Peter Zijlstra pour avoir corrigé l'avertissement concernant lockdep. Merci à Harvey Harrison pour avoir souligné que le unlikely() était inutile.
> Signé par : Matthew Wilcox willy@linux.intel.com (mailto:willy@linux.intel.com)
> Accepté par : Ingo Molnar mingo@elte.hu (mailto:mingo@elte.hu) »[8]

Lorsqu'un code source est développé dans une forge, chaque modification peut être documentée à l'aide d'une telle révision. L'ensemble des révisions permet donc de reconstituer l'historique des changements apportés. Dans l'exemple ci-dessus, Matthew Wilcox justifie sa démarche visant à simplifier une partie du noyau Linux, en remplaçant 7679 lignes complexes par 314 lignes élémentaires, dans 113 fichiers. La révision incarne l'unité intellectuelle qui sous-tend toute cette diversité de modifications ponctuelles. Un historique de développement peut ainsi fournir des informations contextuelles précieuses pour comprendre la structure d'un logiciel.

Enfin, si le code source d'un logiciel est supprimé de son emplacement d'origine, cela n'a pas d'impact sur les fonds de Software Heritage : les évolutions du code source sont archivées mais, si la ressource est supprimée, ce changement n'est pas répercuté – ce qui prémunit contre le risque de lien brisé.

Rendre les logiciels identifiables

Les logiciels constituent des ressources particulièrement difficiles à identifier sur le long terme. Comment surmonter ces difficultés ?

Les besoins d'identification spécifiques aux logiciels

Pourquoi est-il si difficile d'identifier un logiciel ? Certaines raisons sont contextuelles :

- La liste ou le rôle des contributeur·ices d'un logiciel peut évoluer : une personne qui faisait partie des auteur·es principaux de la version 1 peut devenir testeur ou testeuse de la version 2.
- Les logiciels peuvent changer de nom ou de plateforme de développement. Les articles contenant des liens vers des forges ou des sites web classiques deviennent alors des liens brisés.

D'autres raisons sont inhérentes à la nature même du logiciel :

- Les logiciels évoluent de version en version (changements qui peuvent être mineurs comme majeurs), et on peut souhaiter citer une version, voire une étape précise du développement d'une version.
- Les logiciels, même simples en apparence, font intervenir de nombreux composants et leur architecture peut être complexe. Autrement dit, on peut avoir besoin de faire référence uniquement à un élément du logiciel et pas au produit dans son intégralité.

Ainsi, l'archivage des logiciels est une tâche fondamentalement différente de

l'archivage de jeux de données. Les données associées à un article ou un projet scientifique sont généralement dans un état « figé », unique et définitif, dont il suffit d'assurer la disponibilité à long terme, à l'aide d'un entrepôt dédié et d'un identifiant de type DOI. Inversement, le code source est un contenu fortement évolutif : il convient donc de garder la trace de toutes les versions mises à disposition du public.

De plus, chaque production de code informatique, y compris dans le cas des scripts les plus simples, possède généralement une structure sous-jacente très complexe : aucun extrait de code ne peut exister en dehors d'un environnement d'exécution élaboré, devant donc à son tour être archivé et versionné. La nature fondamentalement évolutive et modulaire des logiciels appelle donc à des solutions spécifiques en termes d'archivage et d'identification.

Pourquoi citer les logiciels

Citer un logiciel s'avère nécessaire lorsque celui-ci a joué un rôle déterminant dans la réalisation des travaux de recherche. La question suivante peut servir de repère : dans quelle mesure le logiciel a-t-il un impact direct sur les résultats obtenus ? S'il n'est pas pertinent de citer un outil de traitement de texte, il est par exemple judicieux de citer le logiciel de reconnaissance optique de caractères utilisé. Il est courant de voir cité un article décrivant le logiciel, plutôt que le logiciel lui-même. Or, cette pratique ne permet pas d'identifier facilement et avec certitude le logiciel en question, ainsi que le démontre Mike Jackson :

> « Les auteurs avaient cité un article de l'OGSA-DAI qui semblait impliquer que la version d'OGSA-DAI qu'ils utilisaient était comprise entre 1 et 6. Plus loin dans leur article, les auteurs mentionnaient un composant spécifique aux versions 2.5 à 6. Cependant, les auteurs ont ensuite mentionné un autre composant qui n'était disponible que dans une version totalement différente du logiciel. Sans ma connaissance approfondie du projet OGSA-DAI, il aurait été impossible de déterminer quel logiciel avait été utilisé.[9]. »

Par ailleurs, il n'existe pas toujours d'article associé au logiciel : il peut avoir été développé en dehors d'une communauté académique, ou simplement, n'avoir jamais été présenté dans une revue. Cela renforce donc le besoin de citer le logiciel lui-même, au même titre qu'une autre ressource académique. Les pratiques de citation de logiciels ne sont pas encore très codifiées dans les communautés académiques. Il n'existe pas de standard descriptif. Sur la base de recommandations d'experts, voici une proposition de noyau minimal d'informations à mentionner[10]:

- Le nom du logiciel de la manière la plus précise possible (par exemple le nom du package)
- La date de mise à disposition de la version ou, à défaut, la date d'utilisation du logiciel par l'utilisateur, à l'exemple de ce qui se pratique pour les citations de pages web
- L'auteur du logiciel
- La localisation initiale (par exemple le lien vers la plateforme de développement)
- L'identifiant pérenne (par exemple le SWHID)

Les utilisateur·ices de LaTeX peuvent utiliser le biblatex-software package (https://ctan.org/pkg/biblatex-software) pour faciliter la tâche.

Utiliser des identifiants pérennes adaptés

Les identifiants pérennes sont un moyen d'assurer un accès stable à une ressource, par exemple un document en ligne ou une description de document.

Ils permettent d'établir des liens entre des informations de différentes natures : on sait par exemple qu'un article donné, identifié par un DOI, a été rédigé par telle personne, identifiée à son tour par sa référence ORCID (*Open Researcher or Contributor ID*) (https://orcid.org/). Et cette personne exerce dans une institution elle-même identifiée via le *Research Organization Registry* (ROR) (https://perma.cc/4JFM-NL2K).

Le Comité pour la science ouverte définit ainsi un identifiant pérenne :

> « Numéro ou étiquette alphanumérique, opaque ou explicite, lisible par des machines et par des humains, permettant de désigner et de retrouver de manière univoque et pérenne un objet, un document, une personne, un lieu, un organisme, ou toute entité, dans le monde réel et sur internet.[11] »

Les identifiants réduisent les ambiguïtés. Il peut en effet parfois être difficile de savoir si tel·le auteur·e ayant publié dans une discipline donnée est l'homonyme d'un·e autre auteur·e publiant dans un domaine proche, ou s'il s'agit d'une seule personne active dans plusieurs champs. On peut également citer le cas des institutions, dont le nom peut changer au fil du temps. Plusieurs institutions peuvent fusionner et former une nouvelle entité à décrire. Ces changements rendent difficiles l'identification des ressources produites par ces instances.

On peut également noter que les objets académiques à identifier se sont diversifiés, et que le logiciel fait maintenant partie des ressources vers lesquelles il est nécessaire de créer des liens valables sur le long terme.

Obtenir et utiliser un *SoftWare Hash Identifier* (SWHID)

Comme nous l'avons vu précédemment, même si le DOI est l'identifiant le plus connu et le plus utilisé dans le monde académique, il n'est pas le plus adapté à l'identification des logiciels : on n'a généralement pas de DOI associé à chaque composant du logiciel — encore moins à chaque étape fine de son processus de développement.

Le *SoftWare Hash Identifier* permet de pointer vers différents composants du logiciel, ainsi que vers des actions de son historique de développement. Il en existe cinq types différents, chacun adapté à la façon dont on souhaite faire référence à un logiciel donné. La documentation technique (https://perma.cc/NYR8-C3AL) détaille les types de SWHIDs disponibles. En voici une synthèse :

Type de contenu à identifier	**Type de SWHID à utiliser**
L'accent est mis sur un point de l'historique de développement	*snapshot* (https://archive.softwareheritage.org/swh:1:dir:d198bc9d7a6bcf6db04f476d29314f157507d505;visit=swh:1:snp:c7c108084bc0b *release* (https://archive.softwareheritage.org/swh:1:rel:22ece559cc7cc2364edc5e5593d63 swh:1:rev:309cf2674ee7a0749978cf8265ab91a60aea0f7d)
L'accent est mis sur l'ensemble des contenus du code source à un moment donné	*directory* (https://archive.softwareheritage.org/swh:1:dir:d198bc9d7a6bcf6db04f476d293
L'accent est mis sur une portion précise de code source à un moment donné	*content* (https://archive.softwareheritage.org/swh:1:cnt:94a9ed024d3859793618152ea55

Quel identifiant pour quel besoin ?

Le tableau suivant résume les informations présentées dans cette section.

Besoin	Citer un logiciel	Effectuer des vérifications techniques
Objectif	Attribuer correctement les responsabilités intellectuelles, pour renvoyer vers une version donnée.	S'assurer de la reproductibilité des résultats obtenus avec le logiciel, pour c
Pré-requis	Citer un logiciel nécessite qu'un identifiant pérenne soit attribué à chaque version. L'identifiant doit renvoyer vers des objets et vers leur description générale.	Effectuer des vérifications techniques nécessite qu'un identifiant pérenne r source comme d'explications fournies par les développeurs pour documen de l'objet.
Type d'identifiant	DOI (https://doi.org/10.5281/zenodo.6375528), HAL-ID (https://hal.science/hal-03516539v1)	SWHID (https://archive.softwareheritage.org/swh:1:cnt:353bd74b1b29c30b parmap;visit=swh:1:snp:8ddca416836fbbc2a7704c69db38739bef6b6cae;ar dune-project)
Analogie	La carte d'identité	Les empreintes digitales, l'échantillon d'ADN
Obtention de l'identifiant	Information sur la page du logiciel déposé au préalable sur une plateforme d'archivage (https://zenodo.org/records/6461551), une archive ouverte (https://hal.science/hal-03516539v1)	Le logiciel est archivé dans Software Heritage (https://archive.softwareheri swh:1:dir:bc7ddd62cf3d72ffdc365e1bf2dea6eeaa44e185;origin=https://git parmap;visit=swh:1:snp:8ddca416836fbbc2a7704c69db38739bef6b6cae;ar cliquer sur l'onglet Permalink (https://annex.softwareheritage.org/public/tu

Ces différents identifiants sont complémentaires car ils répondent à des besoins différents. Ainsi, un HAL-ID renvoie vers une description de logiciel incluant un SWHID chargé de pointer vers un contenu donné. Si l'on souhaite renvoyer vers un fichier précis, il faut recourir au SWHID.

En pratique : trouver et référencer des logiciels archivés

Dans cette section, nous voyons comment trouver un logiciel dans l'archive Software Heritage et comment obtenir les identifiants dont vous avez besoin pour le citer correctement dans vos productions académiques. Référencer un logiciel signifie lui attribuer un identifiant pérenne.

Naviguer et rechercher des logiciels sur Software Heritage

Pour consulter ou citer du code source sur Software Heritage, il faut déjà l'avoir trouvé dans l'archive. Mais comment s'orienter dans une archive aussi foisonnante ? Selon le degré d'information que vous possédez sur la ressource recherchée, vous vous trouverez dans l'une des situations suivantes :

1. Vous ne connaissez que le *nom du projet*, par exemple « Linux ».
2. Vous possédez un *fichier du code source*, par exemple le texte de la GNU General Public License v3.
3. Utilisateur avancé, vous connaissez un *extrait du code*, par exemple `unsigned three = 1; .`
4. Vous connaissez l'*adresse du dépôt de référence* du projet, par exemple https://github.com/torvalds/linux (https://github.com/torvalds/linux).
5. Utilisateur avancé, vous possédez un *code de hachage* permettant d'identifier un certain fichier, un répertoire, une version ou une révision.
6. Vous possédez un *SoftWare Hash Identifier* (SWHID).

Cas d'étude : explorer Software Heritage🔗

Examinons chacun des cas présents, en allant du point de départ le plus imprécis vers le plus précis. Les options 1, 2, 4 et 6 permettent une interrogation facile de Software Heritage (Fig. 1).

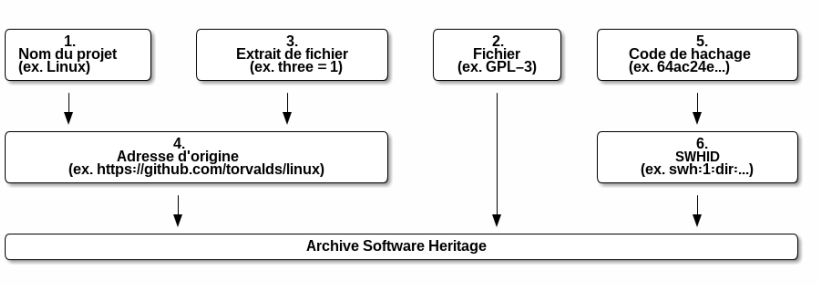

Figure 1. Tous les chemins mènent à Software Heritage.

Option 1. Vous ne connaissez que le *nom du projet*

Pour le projet « Linux », par exemple, le plus simple est de chercher son dépôt officiel dans un moteur de recherche ou un catalogue de logiciels, comme le catalogue des logiciels du MédiaLab de SciencesPo (https://medialab.sciencespo.fr/outils/)). En tapant « Linux source code » dans un moteur de recherche, vous tomberez assez rapidement sur une page intitulée « GitHub - torvalds/linux: Linux kernel source tree » à l'adresse https://github.com/torvalds/linux (https://github.com/torvalds/linux). Cette adresse est celle de l'« origine », c'est-à-dire du dépôt distant de référence. Elle permet de retrouver très facilement le code source sur Software Heritage, comme vous le verrez dans l'Option 4 ci-dessous.

Option 2. Vous possédez un *fichier du code source*

Pour le texte de la GNU Public Licence v3, rendez-vous à l'adresse https://www.softwareheritage.org/ (https://www.softwareheritage.org/). Glissez et déposez votre fichier (vous pouvez le télécharger ici (/assets/preserver-logiciels-recherche/gpl-3.0.txt) pour essayer vous-même cet exercice) dans l'encadré intitulé « Vérifiez si le code source qui vous intéresse est déjà présent dans l'archive », puis cliquez sur *Search* (Recherche). Un menu déroulant vous indiquera alors si le code figure sur Software Heritage et quel est son code de hachage, par exemple `8624bcdae55baeef00cd11d5dfcfa60f68710a02` . Cliquer sur *Browse* (*Parcourir*) vous permettra de visualiser ce fichier à l'adresse https://archive.softwareheritage.org/browse/content/8624bcdae55baeef00cd11d5dfcfa60f68710a02/?path=GPL-3 (https://archive.softwareheritage.org/browse/content/8624bcdae55baeef00cd11d5dfcfa60f68710a02/?path=GPL-3).

Option 3. Vous connaissez un *extrait du code*

Considérons par exemple l'extrait `unsigned three = 1;` . Vous pouvez chercher davantage d'information à son sujet à l'aide d'un moteur de recherche. Jonglez avec les guillemets pour cibler l'expression exacte (comme lorsque vous

effectuez une recherche dans un catalogue ou un moteur de recherche, ajouter des guillemets permet de faire porter la recherche sur l'expression telle quelle), et ajoutez des mots-clés pour affiner les résultats. En tapant par exemple « "unsigned three = 1" source code », vous trouverez rapidement un lien vers la page https://github.com/torvalds/linux/blob/master/fs/ext4/resize.c (https://github.com/torvalds/linux/blob/master/fs/ext4/resize.c), ce qui vous indique l'adresse d'origine du projet et vous conduit une fois de plus à la situation décrite dans l'Option 4 ci-dessous.

Option 4. Vous connaissez l'*adresse du dépôt de référence* d'un projet

Si vous connaissez par exemple l'adresse https://github.com/torvalds/linux (https://github.com/torvalds/linux) du projet Linux, il suffit de la saisir dans la barre de recherche de Software Heritage (https://archive.softwareheritage.org/browse/search/). Vous serez alors automatiquement redirigé·e vers la page correspondante de l'archive (https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/torvalds/linux), qui vous permettra de naviguer dans le code source de Linux. Vous pouvez par exemple explorer le fichier `fs/ext4/resize.c` jusqu'à trouver la ligne contenant l'instruction paradoxale `unsigned three = 1;` .

Option 5. Vous connaissez un *code de hachage*

Vous êtes un utilisateur suffisamment expérimenté et possédez le *code de hachage* (SHA-1) permettant d'identifier un certain fichier, un répertoire, une version ou une révision. Par exemple, en visitant l'adresse https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=64ac24e738823161693bf791f87adc802cf529ff (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=64ac24e738823161693bf791f87adc802cf529ff), vous avez trouvé le numéro de révision `64ac24e738823161693bf791f87adc802cf529ff` , ou bien vous savez que le code de hachage d'un fichier est `8624bcdae55baeef00cd11d5dfcfa60f68710a02` . Vous pouvez alors explorer la page correspondante en rajoutant l'un des préfixes suivants, selon le type de ressource :

- Pour un fichier : `https://archive.softwareheritage.org/browse/content/`
- Pour un répertoire : `https://archive.softwareheritage.org/browse/directory/`
- Pour une capture : `https://archive.softwareheritage.org/browse/snapshot/`
- Pour une version : `https://archive.softwareheritage.org/browse/release/`
- Pour une révision : `https://archive.softwareheritage.org/browse/revision/`

On trouvera donc la révision `64ac24e738823161693bf791f87adc802cf529ff` à l'adresse https://archive.softwareheritage.org/browse/revision/64ac24e738823161693bf791f87adc802cf529ff (https://archive.softwareheritage.org/browse/revision/64ac24e738823161693bf791f87adc802cf529ff), et le fichier dont le code est `8624bcdae55baeef00cd11d5dfcfa60f68710a02` à l'adresse https://archive.softwareheritage.org/browse/content/8624bcdae55baeef00cd11d5dfcfa60f68710a02 (https://archive.softwareheritage.org/browse/content/8624bcdae55baeef00cd11d5dfcfa60f68710a02). Vous pouvez également, pour chacun de ces types de ressources — à l'exception notable des fichiers — obtenir un SoftWare Hash Identifier (SWHID) en ajoutant le préfixe correspondant :

- Pour un répertoire : `swh:1:dir:`
- Pour une capture : `swh:1:snp:`
- Pour une version : `swh:1:rel:`
- Pour une révision : `swh:1:rev:`

Vous vous trouverez alors dans la situation abordée dans l'Option 6 ci-dessous :

Option 6. Vous possédez un identifiant SoftWare Hash Identifier (SWHID)

Vous pouvez alors le taper dans la barre de recherche de l'archive (https://archive.softwareheritage.org/browse/search/). Par exemple, si vous y saisissez l'identifiant « swh:1:dir:1fee702c7e6d14395bbf5ac3598e73bcbf97b030 », vous serez redirigé·e vers la page https://archive.softwareheritage.org/browse/directory/1fee702c7e6d14395bbf5ac3598e73bcbf97b030/ (https://archive.softwareheritage.org/browse/directory/1fee702c7e6d14395bbf5ac3598e73bcbf97b030/).

> À noter : le recours aux catalogues de logiciels reste une pratique peu répandue, y compris parmi les développeur·euses. Pourtant, ces outils spécialisés permettent des recherches plus ciblées que les moteurs généralistes. Vous connaissez les institutions dont les domaines d'expertise sont proches des vôtres : vérifiez s'il n'existe pas un catalogue recensant la production logicielle de cet établissement. Votre propre institution a peut-être développé un tel outil.

Exercice : mettre en œuvre sa stratégie de recherche

Mettons à présent ces notions en pratique. Trouvez dans l'archive Software Heritage :

1. Le code du programme LanguageTool.
2. Le code contenant les mots « you are not expected to understand this » (« vous n'êtes pas supposés comprendre ceci »).
3. Le code correspondant au dépôt https://github.com/CatalaLang/catala (https://github.com/CatalaLang/catala).
4. Le fichier composé des lignes suivantes :

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!\n");
}
```

5. Le répertoire dont le SWHID est `swh:1:dir:f72ab78156c5a4f05d394afa7a2f5911e1f33e27` .

Choisir l'identifiant pérenne à intégrer dans une citation

Maintenant que vous savez naviguer dans l'archive et que vous avez vu ci-dessus les différents types de SWHIDs permettant de répondre aux différents besoins de citation d'un logiciel, donnons ci-dessous deux cas concrets. Dans ces situations, des identifiants pérennes bien choisis sont nécessaires pour répondre aux besoins des chercheurs qui souhaitent faire référence à des logiciels, ou des portions de code source, dans leurs productions académiques.

1. Une doctorante souhaite citer les logiciels utilisés pendant sa thèse pour permettre à ses lecteur·ices d'identifier et retrouver le code source correspondant, et de reproduire les résultats présentés dans son manuscrit. Elle a notamment utilisé le package R `{rdss}` pour une partie de ses analyses statistiques. Comme de nombreux autres packages R, `{rdss}` est seulement disponible sur la forge GitLab.com (https://gitlab.com/f-santos/rdss). Bien que librement téléchargeable, ce package pourrait ne plus être disponible du jour au lendemain, si la forge

GitLab.com fermait soudainement ses portes, ou si l'auteur en retirait son package sans préavis. Les résultats présentés dans la thèse de cette doctorante seraient alors impossibles à auditer et à reproduire par les membres de son jury, ou par les futurs lecteur·ices de son travail. Comment peut-elle fournir un identifiant pérenne pour permettre à ses lecteur·ices d'accéder durablement à la version de `{rdss}` utilisée dans son manuscrit, à savoir la version (ou « release ») 1.1.1 ?

2. Un chercheur rédige un article cherchant à comparer la performance de divers algorithmes de fouille de textes, ou « text mining », une méthode utile notamment en Histoire. Un lot de méthodes de fouille de textes est par exemple implémenté dans le package R `{tm}` (https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://cran.r-project.org/package%3Dtm&snapshot=99204e5a7070f348901b0e966a7ffbbe3db0a9b9&visit_type=cran). Notre chercheur pense avoir développé des fonctions en langage R encore plus rapides et économes en temps de calcul sur de grands jeux de données, en apportant des améliorations décisives à certains endroits du package `{tm}`. Par exemple, en considérant la version 0.7-9 du package `{tm}`, il souhaite signaler qu'il a mis au point une version plus rapide d'une fonction nommée `tm_scan_one`, dont le code est situé dans le fichier `src/scan.c` (https://archive.softwareheritage.org/browse/content/sha1_git:e76e4f8b6a1d34dcb55cebaa0f4b91e5a186dd08/?origin_url=https://cran.r-project.org/package%3Dtm&path=tm/src/scan.c&snapshot=99204e5a7070f348901b0e966a7ffbbe3db0a9b9&visit_type=cran) du package. En particulier, ce chercheur a apporté une amélioration décisive à la portion de code comprise entre les lignes 46 à 65, écrite en langage C. Comment peut-il faire référence de manière pérenne à cette portion de code dans son projet de publication ?

Solutions

1. Le but est ici de citer une version d'un logiciel : le bon identifiant SWHID est donc un identifiant portant sur la « release ». Sur la page adéquate (https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://gitlab.com/f-santos/rdss) de l'archive Software Heritage, on peut sélectionner la release 1.1.1, puis obtenir l'identifiant de version `swh:1:rel:ef8ba743282a602d8b105ce82e1f6f48779d7998` grâce au menu **Permalinks** situé à droite de l'écran (Fig. 2).

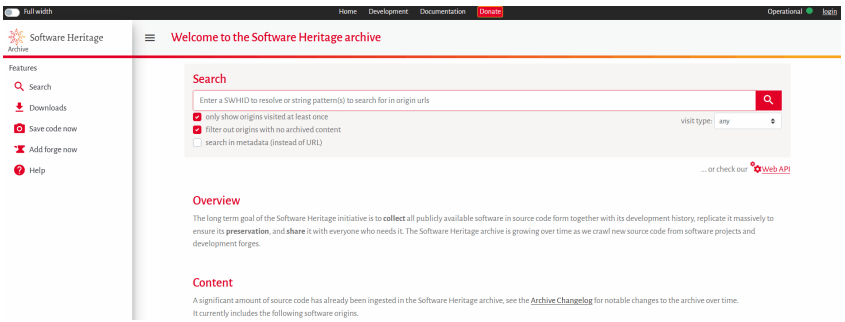

Figure 2. Trouver et citer une version précise d'un package R dans l'archive Software Heritage.

2. Le but est ici de citer un extrait de code contenu dans un fichier source, à une étape donnée du développement d'un logiciel. Le bon identifiant SWHID est donc un identifiant de type « content », c'est-à-dire, portant sur le contenu d'un fichier précis. Sur la page adéquate (https://archive.softwareheritage.org/browse/content/sha1_git:e76e4f8b6a1d34dcb55cebaa0f4b91e5a186dd08/?origin_url=https://cran.r-project.org/package=tm&path=tm/src/scan.c&snapshot=99204e5a7070f348901b0e966a7ffbbe3db0a9b9&visit_type=cran#L46-

L65) de l'archive Software Heritage, on peut sélectionner la release 0.7-9 du package, puis les lignes 46 à 65, pour enfin obtenir l'identifiant de contenu `swh:1:cnt:e76e4f8b6a1d34dcb55cebaa0f4b91e5a186dd08` grâce au menu **Permalinks** situé à droite de l'écran (Fig. 3).

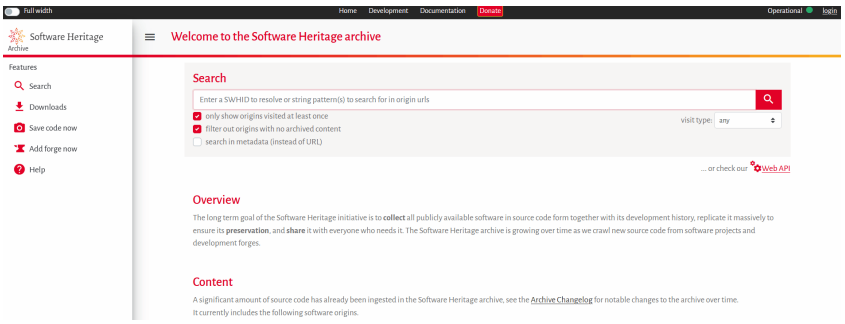

Figure 3. Trouver et citer un extrait de code source précis dans l'archive Software Heritage.

Archiver du code source🔗

Dans cette section, vous allez principalement vous placer du point de vue des producteurs de code source, en créant un dépôt contenant un bref extrait de code et en assurant son archivage sur Software Heritage. Toutefois, rappelons qu'il n'est pas nécessaire d'être l'auteur·e d'un logiciel pour en demander l'archivage sur Software Heritage.

Dans quels cas archiver manuellement du code source ?🔗

La grande majorité du code source archivé sur Software Heritage provient des « moissons » automatiques effectuées périodiquement depuis différentes sources. Notons que ces moissons automatiques sont faites « sans filtre » : Software Heritage n'effectue pas de test technique pour vérifier si un logiciel fonctionne.

Entre ces moissons périodiques, tout·e auteur·e ou utilisateur·ice de logiciel a la possibilité de déclencher manuellement l'archivage d'un nouveau dépôt, ou la mise à jour d'un dépôt existant. Il s'agit de l'option *Save Code Now* (https://archive.softwareheritage.org/save/) (*Sauvegarder le Code Maintenant*), permettant de soumettre l'URL d'un dépôt de code source qui sera alors inspecté puis archivé par Software Heritage dans son état le plus récent. Il existe au moins deux cas où vous pourriez souhaiter utiliser cette option :

1. Vous êtes utilisateur·ice d'un logiciel et avez utilisé sa toute dernière version pour produire les résultats d'un article que vous vous préparez à soumettre. Vous souhaitez utiliser un SWHID judicieusement choisi (en l'occurrence, un SWHID de type `release`) pour citer ce logiciel dans votre article. Malheureusement, la dernière version de ce logiciel n'a pas encore été moissonnée automatiquement par Software Heritage, et le SWHID correspondant ne peut donc pas être encore être produit. Vous souhaitez donc utiliser l'option *Save Code Now* afin que Software Heritage visite à nouveau le dépôt de ce logiciel, et en archive la dernière version. Le SWHID dont vous avez besoin sera alors disponible.
2. Vous êtes auteur·e d'un logiciel développé sur la forge GitLab et vous souhaitez le publier rapidement. Vous ne pouvez pas attendre que GitLab.com soit moissonné automatiquement et souhaitez en déclencher l'archivage tout de suite afin de pouvoir fournir une URL stable (et un SWHID) dans l'article de présentation de votre logiciel.

Étudions concrètement le premier de ces deux cas, du point de vue d'un·e utilisateur·ice non-développeur·se.

Vérifier que la version archivée d'un dépôt est à jour🔗

À un instant donné, il se peut que la version d'un logiciel actuellement disponible sur l'archive Software Heritage corresponde à une version antérieure à celle disponible sur la source officielle. Cela est particulièrement susceptible de se produire pour les logiciels en développement très actif dont l'évolution est rapide, avec parfois plusieurs dizaines de nouvelles révisions par jour. En effet, Software Heritage est une archive et non une forge, et n'a donc pas vocation à offrir en temps réel les dernières versions de l'ensemble des logiciels référencés. On peut néanmoins vérifier aisément si la version disponible sur l'archive Software Heritage est à jour.

Prenons l'exemple d'un logiciel libre développé collaborativement sur la forge logicielle GitHub : le package `citar` pour l'éditeur de texte Emacs (https://www.gnu.org/software/emacs/). En naviguant sur le dépôt correspondant de l'archive Software Heritage (Fig. 4), on peut repérer la date du dernier « snapshot » (capture) effectué sur le dépôt GitHub (ici encadrée en bleu), ainsi que le « hash » (c'est-à-dire l'identifiant) de la dernière modification connue sur ce dépôt (ici encadré en vert). En comparant ces éléments avec la date et le hash de la dernière modification effectuée sur le dépôt GitHub d'origine (https://github.com/emacs-citar/citar), on peut donc savoir si la version archivée est à jour ou non.

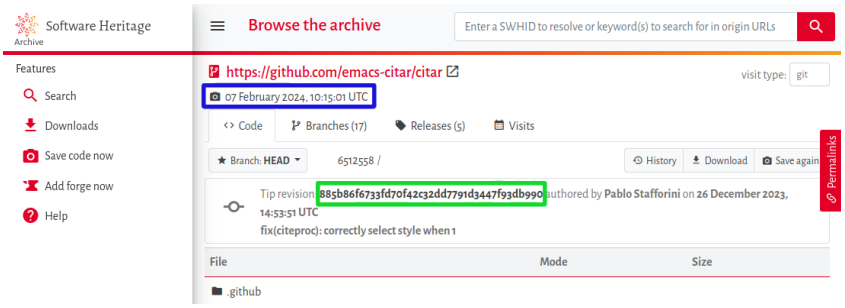

Figure 4. Capture d'écran de la version archivée du package Emacs `citar`.

Notons qu'une extension de navigateur, UpdateSWH (https://www.softwareheritage.org/browser-extensions/), disponible pour les principaux navigateurs web, facilite cette opération. Après l'avoir installée, il suffit de visiter le dépôt d'origine d'un logiciel sur une forge moissonnée par Software Heritage. Une icône s'affiche alors à droite de l'écran, accompagnée d'une infobulle précisant si la version de la forge de développement coïncide ou non avec celle de Software Heritage (Fig. 5). Ici, la dernière version de `citar` disponible sur GitHub est plus récente que celle archivée sur Software Heritage : les modifications les plus récentes n'ont donc pas encore été archivées.

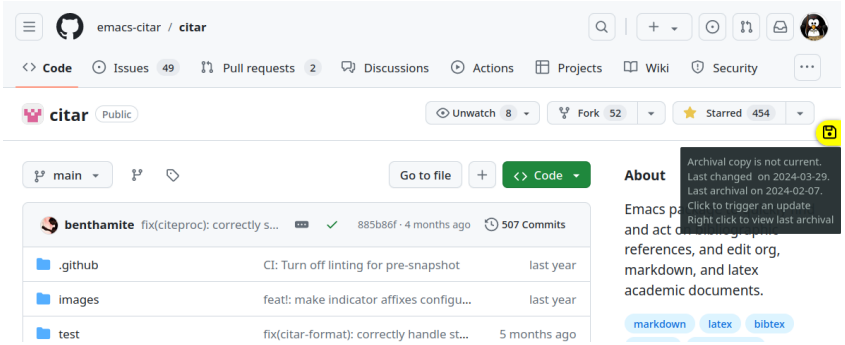

Figure 5. Capture d'écran du dépôt GitHub du package Emacs `citar`. Le plug-in de navigateur Software Heritage fournit des informations dans l'infobulle à droite.

En pratique : archiver du code source que vous avez écrit

Afin de reprendre le cas d'usage le plus fréquent, vous allez créer un dépôt sur

une forge logicielle basée sur Git (https://git-scm.com/), avant d'y éditer des extraits de code. Git est un logiciel de versionnement avancé, permettant de gérer finement les versions successives de fichiers (au format texte brut principalement) d'un même projet, lors de l'avancée du travail. Il permet de garder la trace de chaque modification effectuée (date, auteur·e, contenu) et de pouvoir revenir à des versions antérieures de ces fichiers. Il existe d'autres systèmes de gestion de version (par exemple Mercurial (https://www.mercurial-scm.org/) ou SVN (https://subversion.apache.org/)), mais la plupart des forges logicielles actuelles utilisent Git, et offrent ainsi une plateforme web centralisant l'avancée d'un projet, en particulier dans le cas d'un développement collaboratif.

> Vous n'avez pas besoin d'avoir déjà utilisé Git pour faire les exercices. Nous proposons la création d'un compte sur la forge logicielle GitLab.com, et vous utiliserez l'éditeur de code en ligne directement intégré sur cette forge.
> </br> Si vous êtes au contraire déjà un·e utilisateur·ice régulier·e de Git, vous pouvez effectuer l'exercice en utilisant votre compte sur une autre forge logicielle (par exemple GitHub, ou une forge institutionnelle), et en utilisant Git en ligne de commande avec les instructions `add`, `commit`, `push`, etc., si vous en avez l'habitude.

Créer un dépôt GitLab

1. Si ce n'est pas déjà fait, commencez par vous créer un compte sur GitLab.com (https://gitlab.com/users/sign_up).
2. Vous pourrez ensuite vous connecter sur cette forge (https://gitlab.com/users/sign_in) avec vos identifiants.
3. Dans l'interface de GitLab.com, ou en suivant ce lien (https://gitlab.com/projects/new), choisissez l'option *Créer un projet vide*, dans lequel vous hébergerez par la suite du code source.
4. Choisissez par exemple `Test SWH` comme nom de projet, et veillez à ce que le **Niveau de visibilité** du projet soit réglé sur « Public ». (Sans cela, le code source du projet ne serait accessible que par vous, et ne pourrait donc pas être moissonné par Software Heritage.)
5. Validez enfin la création de ce nouveau projet en cliquant sur le bouton *Créer le projet* (Fig. 6).

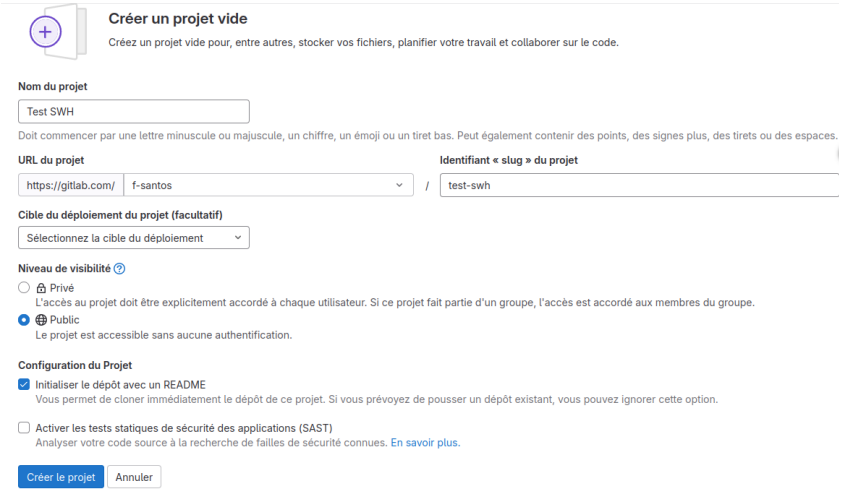

Figure 6. Créer un nouveau projet dans l'interface de GitLab.com.

Écrire une portion de code

Vous devriez alors arriver sur la page du nouveau projet `Test SWH` ainsi créé. Ce projet est pour l'instant vide, à l'exception d'un fichier `README.md` prérempli.

Vous pouvez à présent insérer un extrait de code dans votre projet. Par

exemple, vous pouvez réutiliser l'extrait de code suivant (en langage Bash), prenant en argument une année, et déterminant si cette année est bissextile ou non.

```
#!/usr/bin/env bash

# Check arguments:
if [[ $# -ne 1 || ! $1 =~ ^[0-9]*$ ]]
then
    echo "Usage: leap.sh <year>"
    exit 1
fi

# Do we have a leap year?
if (($1 % 400 == 0)) || ( (($1 % 4 == 0)) && (($1 % 100 != 0)) )
then
    echo "true"
else
    echo "false"
fi

exit 0
```

Pour ce faire, dans l'interface de GitLab.com, cliquez sur le bouton `+` situé en haut de la page du dépôt, puis choisissez **Nouveau fichier**. Dans la nouvelle fenêtre qui s'ouvre, collez l'extrait de code ci-dessus, puis attribuez le nom `leap.sh` à ce fichier source dans le champ Filename. Enfin, en bas de la page, cliquez sur *Valider les modifications* pour effectuer une révision. Le fichier `leap.sh` est maintenant ajouté à votre dépôt, et une révision associée à la création de ce fichier a bien été enregistrée (Fig. 7).

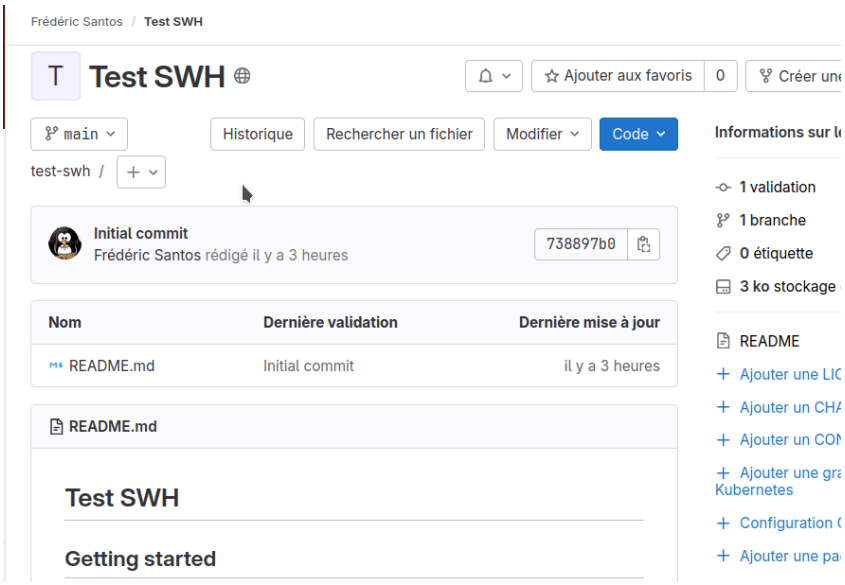

Figure 7. Ajout du fichier `leap.sh` dans votre dépôt.

Ajouter une licence

Maintenant que votre dépôt comporte du code, il est nécessaire de lui ajouter une licence pour en spécifier les conditions d'utilisation. Certains sites spécialisés, tels que Choosealicense.com (https://choosealicense.com/), pourront vous aider à choisir la licence adaptée à vos besoins et au degré de liberté que vous souhaitez accorder aux utilisateurs de votre logiciel.

D'un point de vue technique, on peut aisément attribuer une licence à travers l'interface de GitLab.com, en cliquant sur le lien *Ajouter une LICENCE* dans la page d'accueil du dépôt. Vous pouvez alors choisir l'une des licences dans la liste déroulante **Appliquer un modèle**. Choisissez par exemple la licence GNU

Affero General Public License v3.0, puis cliquez à nouveau sur *Valider les modifications* en bas de la page. Cela crée une nouvelle révision associée à l'ajout du fichier `LICENSE` dans votre dépôt (Fig. 8).

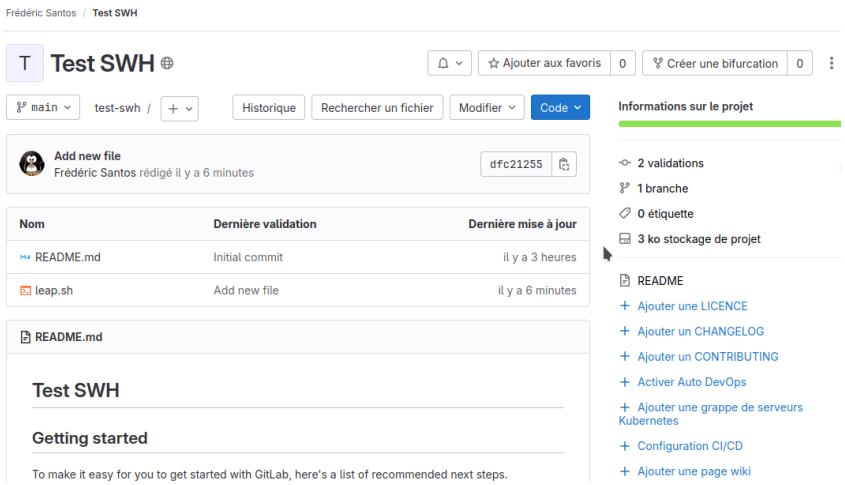

Figure 8. Ajout d'une licence.

Déclencher l'archivage du dépôt

Votre dépôt GitLab possède désormais un contenu raisonnable : un fichier de code en langage Bash, une licence spécifiant les conditions d'utilisation, et un « template » (modèle exemple) de fichier README. Ce dépôt peut à présent être archivé sur Software Heritage. Deux options sont possibles pour cela :

1. Manuellement, avec l'option *Save Code Now*. Visitez la page dédiée (https://archive.softwareheritage.org/save/) sur le site de Software Heritage. Entrez l'URL de votre dépôt `Test SWH` créé sur GitLab.com. Notons que, par défaut, le champ **Origin type** est correctement attribué, avec la valeur `git` (qui correspond effectivement au cas d'un dépôt GitLab). Collez dans le champ **Origin URL** l'adresse que vous pouvez récupérer sur la page GitLab de votre dépôt, en choisissant « Cloner avec HTTPS » dans le menu déroulant **Code** (Fig. 9).

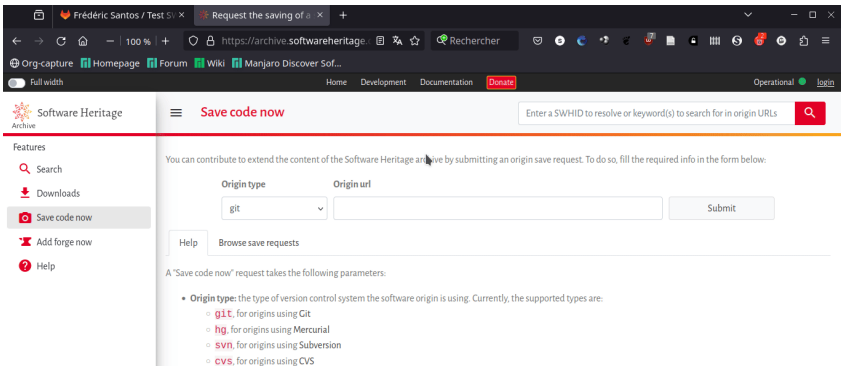

Figure 9. Archiver manuellement un dépôt.

2. Alternative : vous pouvez également choisir d'utiliser le plug-in de navigateur (https://www.softwareheritage.org/browser-extensions/) déjà mentionné plus haut. Cliquez sur l'icône « disquette » affichée à droite de votre écran afin de déclencher l'archivage (Fig. 10).

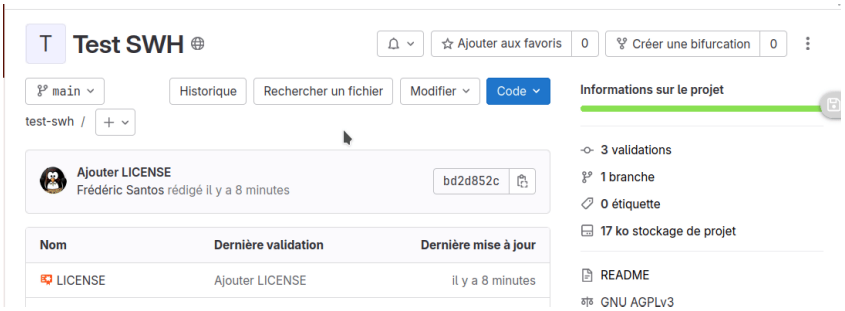

Figure 10. Archiver un dépôt à l'aide du plug-in de navigateur.

Pour aller plus loin : décrire un logiciel avec CodeMeta

Fournir un fichier README (https://perma.cc/C6UN-STC3) suffisamment détaillé (voir par exemple le fichier README du package R `{tapas}` (https://archive.softwareheritage.org/swh:1:cnt:0918fd32b5eddc85aaaeb35da83a749795e703bc;origin=https://github.com/wfinsinger/tapas;visit=swh:1:snp:fb13c70f7d5edc7515756d6f1c01a3f3b1a69e72;anchor=swh:1:rev:5f3d58423904da1README.Rmd)) constitue une étape importante pour expliquer les finalités d'un logiciel. Néanmoins, ajouter un fichier descriptif CodeMeta facilite l'identification d'un logiciel.

Ajouter des métadonnées pour rendre son logiciel identifiable

Les logiciels sont des ressources changeantes : leur cycle de vie peut s'étendre sur plusieurs décennies, leurs fonctionnalités évoluent, leur nom n'est pas toujours fixe, leur plateforme d'origine peut aussi changer... Cela, y compris en cours du développement d'une même version.

De fait, ces ressources peuvent être difficiles à identifier. Dans le cas des logiciels, les métadonnées doivent entre autres permettre d'exprimer les auteur·es, les changements de versions, la ou les licences, mais aussi les relations entre les différentes productions académiques. Les métadonnées aident à identifier le logiciel ainsi que son contexte de création et d'utilisation. Il existe deux grandes catégories de métadonnées.

1. Les métadonnées intrinsèques fournissent des informations essentielles pour assurer la préservation et l'utilisation à long terme des logiciels. Elles sont contenues dans un fichier texte à vocation descriptive, souvent intitulé `README` ou `DESCRIPTION` , situé à la racine du répertoire du code source du logiciel. C'est pourquoi, très souvent, les auteur·es de logiciels sont les premiers et les principaux fournisseurs de métadonnées intrinsèques.

2. Les métadonnées extrinsèques peuvent fournir quant à elles des informations importantes sur le contexte et la provenance du logiciel : par exemple, sur quelle forge le logiciel a été développé. Ces informations aident les utilisateurs à localiser le logiciel, et à identifier la communauté qui l'utilise. Ces métadonnées renseignent aussi sur la relation entre le logiciel et d'autres produits de recherche : des publications, des jeux de données... Ces métadonnées ne sont pas incluses dans les fichiers du code source, c'est pourquoi elles sont qualifiées d'extrinsèques. Elles peuvent être ajoutées par des opérateurs extérieurs : des services éditoriaux, des agrégateurs, des catalogues (https://zbmath.org/software/), etc. Elles peuvent être aussi fournies par la personne qui a écrit le code source, comme c'est le cas avec l'archive ouverte HAL. Les informations à compléter sont alors indiquées dans un formulaire de saisie (https://perma.cc/XQ73-HVGK).

La personne à l'origine du logiciel joue un rôle majeur dans le processus de description du logiciel. Des métadonnées riches contribuent à la réutilisation et l'identification efficaces des logiciels.

Les avantages de CodeMeta

Fournir des métadonnées intrinsèques est la responsabilité de l'auteur·e du logiciel. Malheureusement, décrire un logiciel est une pratique bien moins codifiée que décrire un ouvrage ou un article. Si certaines métadonnées semblent évidentes (nom du logiciel, auteur·es, version, licence), il n'existe pas de standard indiquant un noyau minimal d'informations à fournir. La liste des métadonnées dépend de l'usage ciblé, ainsi que le note Sheila M. Morrissey :

> « Comment décrire efficacement un logiciel ? La liste des propriétés à décrire dépend des objectifs pour lesquels nous collectons des logiciels.[12]. »

De plus, il existe de nombreux langages pour décrire ces propriétés, ayant chacun leur vocabulaire spécifique. Non seulement il faut déterminer les éléments à décrire, mais il faut également considérer avec quel vocabulaire les exprimer. Voici un exemple de correspondances entre métadonnées issues de différentes plateformes :

Métadonnée	**CodeMeta**	**GitHub**	**PyPI**
Langage de programmation	programmingLanguage	languages_url	classifiers['Programming Language']
Auteur·e	author	login	author

De même, cette fois à l'échelle du logiciel et non plus de la plateforme, comparez le fichier `DESCRIPTION` d'un package R (https://archive.softwareheritage.org/browse/content/sha1_git:29c0f87d31f0dad1b55bb6d9fed4e7d45b77b19d/?origin_url=https://cran.r-project.org/package%3Dtm&path=tm/DESCRIPTION&snapshot=99204e5a7070f348901b0e966a7ffbbe3db0a9b9&visit_type=cran) et le fichier `Project.toml` d'un package Julia (https://archive.softwareheritage.org/browse/content/sha1_git:4869ca4be8d0b31d1f41ee164b658f0c617a8030/?origin_url=https://github.com/JuliaGeometry/Meshes.jl&path=Project.toml). Ces deux fichiers ont le même objectif (indiquer auteur·es, versions, dépendances, etc.), mais vous pouvez constater que leur syntaxe diffère sensiblement en fonction du langage, ce qui rend difficile leur moisson et leur traitement automatiques.

CodeMeta résout ces problèmes en jouant le rôle de connecteur entre une vingtaine de vocabulaires (https://codemeta.github.io/crosswalk/) : il est comparable à une table de concordance. CodeMeta a aussi l'avantage d'être adapté aux besoins académiques. Il permet ainsi d'indiquer un financement, un lien avec une publication, ou encore un domaine disciplinaire. Les métadonnées sont lisibles par l'humain comme par la machine, ce qui augmente le potentiel de « découvrabilité » d'un logiciel décrit via un fichier CodeMeta, conçu avec le CodeMeta Generator.

En pratique : utiliser CodeMeta Generator pour documenter votre dépôt

Ajouter un fichier CodeMeta dans le dépôt de la leçon permet de renseigner quelques métadonnées utiles sous un format unifié. Ce fichier, nommé `codemeta.json`, est donc au format JSON (JavaScript Object Notation) (https://perma.cc/BY6X-HDJN), l'un des formats populaires de stockage de (méta)données. Cette syntaxe est néanmoins très fastidieuse à saisir manuellement, et on utilise donc plutôt habituellement des interfaces de saisie pour générer de tels fichiers.

En effet, consultez par exemple ce fichier CodeMeta (https://archive.softwareheritage.org/

swh:1:cnt:3f28ad2b83ae59ff2ab85c95aab666f0dca8487b;origin=https://github.com/damienbelveze/plagiator;visit=swh:1:snp:3dd49cc39bde69ed9fc8819a745ff0441cf86495;anchor=swh:1:rev:c89473931599codemeta.json), fournissant les métadonnées pour le « serious game » (jeu sérieux) Plagiator. Bien que l'on puisse comprendre la structure globale d'un tel fichier, et deviner le sens de la plupart des champs (`contributor` , `email` , `dateCreated` , etc.), il semble clair que la saisie manuelle de ce genre de fichier serait inconfortable pour l'auteur·e du logiciel.

Vous pourrez donc préférer utiliser des interfaces graphiques intuitives telles que CodeMeta Generator (https://codemeta.github.io/codemeta-generator/) pour produire des fichiers CodeMeta.

Exercice. Créez un fichier CodeMeta correspondant à votre dépôt.

1. Sur le site de CodeMeta Generator (https://codemeta.github.io/codemeta-generator/), remplissez au moins les champs les plus essentiels (par exemple : `Name` , `Authors` , `License(s)` , `Programming Language` , `Code repository`) avec les informations adéquates.
2. Cliquez sur le bouton *Generate codemeta.json v3.0* (*Générer codemeta.json v3.0*) en bas de la page afin d'obtenir le contenu d'un fichier `codemeta.json` .

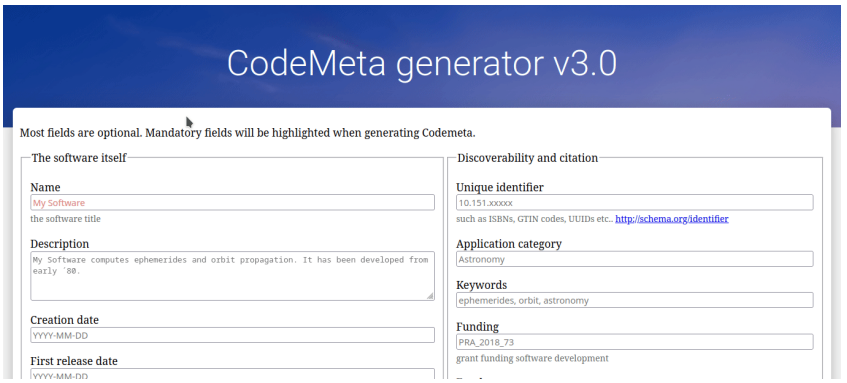

Figure 11. Créer un fichier CodeMeta.

Exercice. Placez ce fichier `codemeta.json` à la racine de votre dépôt.

1. Copiez le contenu proposé par CodeMeta Generator à l'issue de l'étape précédente.
2. Dans votre dépôt sur GitLab.com, créez un nouveau fichier nommé `codemeta.json` , et collez-y ce contenu.
3. Cliquez sur *Valider les modifications* afin de créer une nouvelle révision (« commit ») associée à la création de ce fichier.

Figure 12. Ajouter le fichier CodeMeta à votre dépôt.

Conclusion🔗

Une recherche plus transparente et plus reproductible nécessite non seulement un accès pérenne aux jeux de données, mais également aux logiciels utilisés dans les travaux académiques. Par leur structure, leurs modalités de conception et de diffusion, les logiciels doivent être archivés avec des solutions spécifiques telles que la bibliothèque de code source Software Heritage. En effet, si les forges constituent de puissants outils de développement, elles n'offrent aucune garantie de disponibilité à long terme de leur contenu. De plus, grâce aux différentes types d'identifiants pérennes, il est possible de garantir l'accès aux logiciels et à leurs composants de manière fiable. Enfin, que le logiciel soit composé de millions de lignes de code ou qu'il soit un court script, il représente une production académique méritant d'être archivée et citée.

Les pratiques académiques relatant aux logiciels sont encore en plein développement. Intégrer dès maintenant un socle de bonnes pratiques vous bénéficiera sur la durée.

Avez-vous plus de questions ?

Les auteur·es se feront un plaisir de répondre à toute question que vous pouvez avoir.

- Pour toute demande relative à Software Heritage, consultez la foire aux questions (https://perma.cc/FEL3-4MP6) ou utilisez le formulaire de contact (https://www.softwareheritage.org/contact/).
- Pour suivre l'actualité de Software Heritage, consultez le blog dédié (https://www.softwareheritage.org/blog/) ou abonnez-vous à la newsletter (https://www.softwareheritage.org/newsletter/).

Remerciements

Les auteur·es souhaitent remercier Joenio Marques da Costa, « backend developer » de la plateforme CorTexT (https://www.cortext.net/).

Notes de fin

1. Notre traduction. Texte original :« A common theme is a growing emphasis on "reproducibility" being discussed in many disciplines [...]. This goes beyond "data" and requires software and analysis pipelines to be published in a usable state alongside papers. ». Davenport, James Harold, James Grant, et Catherine Mary Jones. « Data Without Software Are Just Numbers ». Data Science Journal 19, nº 1 (22 janvier 2020): 3. https://doi.org/10.5334/dsj-2020-003 (http://datascience.codata.org/articles/10.5334/dsj-2020-003/galley/929/download/) (https://doi.org/10.5334/dsj-2020-003). ↩

2. Le lecteur confronté à un lien brisé peut certes tenter de rechercher et d'obtenir le logiciel mentionné par d'autres moyens, mais il sera probablement difficile d'obtenir la même version logicielle que celle vers laquelle pointait le lien désormais mort. ↩

3. Spinellis, Diomidis, « The decay and failures of web references », Communications of the ACM, janvier 2003, vol. 46, nº 1, p. 71-77, https://doi.org/10.1145/602421.602422 (https://dl.acm.org/doi/pdf/10.1145/602421.602422) (https://doi.org/10.1145/602421.602422). ↩

4. Une forge est une plateforme en ligne permettant à plusieurs auteur·es de travailler ensemble à la production de fichiers au format texte, le cas le plus fréquent (mais pas unique) étant la production de code informatique. Nous reviendrons plus tard dans cette leçon sur la notion de forge logicielle. ↩

5. Escamilla, Emily, Martin Klein, Talya Cooper, Vicky Rampin, Michele C. Weigle, et Michael L. Nelson. « The Rise of GitHub in Scholarly

Publications ». arXiv, 9 août 2022. https://doi.org/10.48550/arXiv.2208.04895 (https://doi.org/10.48550/arXiv.2208.04895). ↩

6. Formellement, notons qu'il existe des langages « compilés » (C/C++, Fortran, ...) dont le code nécessite d'être compilé globalement par le développeur afin de produire un programme exécutable, et des langages « interprétés » (Python, R, ...) dont le code est traduit à la demande (ligne par ligne) dans un format lisible par la machine, à chaque exécution et sans produire de fichier exécutable séparé. Bien que différentes, les étapes de compilation et d'interprétation traduisent toutes deux le code source lisible par des humains en instructions de plus bas niveau exécutables par la machine. ↩

7. Techniquement, Software Heritage peut archiver des exécutables (https://perma.cc/TG2J-V3HS), mais insistons sur le fait qu'il ne s'agit alors que d'un effet secondaire dans l'hébergement du logiciel dans son ensemble. La priorité est la collecte du code source. ↩

8. Notre traduction. Texte original : « Generic semaphore implementation. Semaphores are no longer performance-critical, so a generic C implementation is better for maintainability, debuggability and extensibility. Thanks to Peter Zijlstra for fixing the lockdep warning. Thanks to Harvey Harrison for pointing out that the unlikely() was unnecessary.Signed-off-by: Matthew Wilcox willy@linux.intel.com (mailto:willy@linux.intel.com). Acked-by: Ingo Molnar mingo@elte.hu (mailto:mingo@elte.hu) » ↩

9. Notre traduction. Texte original :« The authors had cited an OGSA-DAI paper that should have meant they were using a version of the software between OGSA-DAI 1 and 6. Later in their paper, the authors mentioned a component that was specific to OGSA-DAI versions 2.5 to 6. However, the authors then talked of another component and a toolkit, which was only available with a completely different version of the software. Without my highly detailed knowledge of the OGSA-DAI project, it would have been impossible to determine what software was used. ». Jackson, Mike. « How to Cite and Describe Software ». Software Sustainability Institute, s.d. https://www.software.ac.uk/how-cite-software (https://perma.cc/U9C5-2W9X). ↩

10. Chue Hong, Neil P., Alice Allen, Alejandra Gonzalez-Beltran, Anita de Waard, Arfon M. Smith, Carly Robinson, Catherine Jones, et al. « Software Citation Checklist for Authors (0.9.0) ». FORCE11 Software Citation Implementation Working Group, 15 octobre 2019. https://doi.org/10.5281/zenodo.3479199. Katz, Daniel S, Daina Bouquin, Neil P Chue Hong, Jessica Hausman, Daniel Chivvis, Tim Clark, Mercè Crosas, et al. « Software Citation Implementation Challenges », 2019, 26. https://doi.org/10.48550/arXiv.1905.08674 (https://doi.org/10.48550/arXiv.1905.08674). ↩

11. Collège Europe Et International et Comité pour la science ouverte. « Des identifiants ouverts pour la science ouverte ». Report. Comité pour la science ouverte, 3 juillet 2019. https://doi.org/10.52949/22 (https://hal-lara.archives-ouvertes.fr/hal-03640303/document) (https://doi.org/10.52949/22). ↩

12. Notre traduction. Texte original :« How do we describe software effectively ? The list of properties to be described will vary according to the purposes for which we collect software. ». Morrissey, Sheila M. « Preserving Software: Motivations, Challenges and Approaches ». Digital Preservation Coalition, août 2020. https://doi.org/10.7207/twgn20-02 (https://doi.org/10.7207/twgn20-02) (https://doi.org/10.7207/

twgn20-02). ↩

À PROPOS DES AUTEUR(E)S

Sabrina Granger est open science community manager au sein de Software Heritage, une infrastructure dédiée à la préservation et au partage de code source. (https://orcid.org/0000-0002-9081-3851)

Baptiste Mélès est chercheur au CNRS (AHP-PReST, Université de Lorraine). Spécialiste de philosophie de l'informatique, il a cofondé le séminaire Codes Sources. (https://orcid.org/0000-0002-5692-083X)

Frédéric Santos est ingénieur d'études en statistique au laboratoire PACEA (De la Préhistoire à l'Actuel : Culture, Environnement, Anthropologie), situé à Bordeaux. (https://orcid.org/0000-0003-1445-3871)

POUR CITER

Sabrina Granger, Baptiste Mélès, et Frédéric Santos, « Préserver et rendre identifiables les logiciels de recherche avec Software Heritage », *Programming Historian en français* 6 (2024), https://doi.org/10.46430/phfr0034. (https://apis.ebsco.com/public/linkout/v1/ftf?ref=2290bb27-6287-4def-a7e7-169a61e65b87&id=229748)

ISSN 2397-2068 (anglais) (/)

Site hébergé sur Github (https://github.com/programminghistorian/jekyll)

ISSN 2517-5769 (espagnol) (/es)

ISSN 2631-9462 (français) (/fr)

ISSN 2753-9296 (portugais) (/pt)

Dernière mise à jour le 08 January 2025 (https://github.com/programminghistorian/jekyll/commits/gh-pages)