

Automating Software Citation using GitCite

Leshang Chen and Susan B. Davidson
 Dept. Computer and Information Science
 University of Pennsylvania
 {leshangc, susan}@seas.upenn.edu

Abstract—The ability to cite software and give credit to its authors and contributors is increasingly important. While the number of online open-source software repositories has grown rapidly over the past few years, few are being properly cited when used due to the difficulty of creating appropriate citations and the lack of automated techniques. This paper presents GitCite, a model for software citation with version control which enables citations to be inferred for any project component based on a small number of explicit citations attached to subdirectories/files, and an implementation that integrates with Git and GitHub. The implementation includes a browser extension and a local executable tool, which enable citations to be added/modified/deleted to software project repositories and managed through functions such as fork/merge/copy.

I. INTRODUCTION

As software becomes an increasingly important research product in Big Data-driven science, appropriate citation is essential to motivate members of the community to continue to contribute as well as to enable reproducibility (see e.g., [1]). Influential statements on the importance of software citation have been published within the digital libraries community (e.g. FORCE 11 reports [2], [3]). However, despite widespread agreement about the importance of these standards, their complexity and the effort entailed discourages users from following them when generating citations by hand. Systems must therefore be developed to automate citations and deliver them along with the software component being used.

To address this problem, several approaches have emerged. First, citations may be explicitly embedded in the comment section of the code in a repository by project members. This approach requires significant effort on the part of both project members and users of the code. Second, a released version of a software project may be treated as open-access data and uploaded to a public hosting platform (e.g., Zenodo [4]) which provides a DOI, thus enabling a traditional form of citation and ensuring persistence. This approach still requires a citation to be constructed for the release version. Third, plug-ins can be developed to take a code source URL as input and return a citation for the whole project as a string. This approach reduces effort on the part of both project members and users since metadata is automatically extracted from the repository to construct the citation, which is then automatically returned to the user (e.g., the Software Citation Tool [5]).

However, there are several important aspects of software citation that none of the current approaches address, which we discuss next.

Granularity: A software project has many different components. It may include different versions, each of which is a large directory/tree structure. Different project versions may be managed and contributed to by different people, and different subtrees of a project version may involve different contributors. Therefore, the citation to a component of a project (e.g. a file) may be different from that for the whole project since the contributors are different; there is a need for *fine-grained* citations.

Modularity: While the first approach enables fine-grained citations, explicitly attaching a citation to the comment section of every file is cumbersome. There may also be a lot of repeated information, since source code files in the same folder typically share common metadata. Therefore, the approach should scale to large repositories by reducing user effort and sharing common metadata across citations when possible.

Usability: Since project repositories are constantly evolving, a software citation system should enable citations to be updated as changes are made to files. The current approaches are static, and most treat source code as a published snapshot or webpage. Integrating a software citation tool within a (heavily used) version control system such as Git will allow citations to be versioned along with the code, and carried through known operators such as fork. Citations may also be modified as contributions to the same file in different versions change over time.

Example. To illustrate the need for fine-grained citations, we use as an example a project called B on GitHub that is owned by Bob. The B project imports a component called CC from a code repository called C owned by Carlos, and modifies CC to dovetail with other parts of the project. Although a comment appears in the readme file for CC giving credit to Carlos, it is not part of the metadata for the file and can easily be overlooked. If another user uses CC and cites the project repository as a whole, credit would wrongly be given to Bob. The B project code is then branched to enable another contributor, Alice, to independently develop a new component in a separate directory, which is later merged with the main branch of code development. Although the blame history reveals that Alice creates and edits files between versions, this can again be easily overlooked and credit incorrectly given to the repository owner Bob. □

Challenges. There are tensions between the granularity, modularity and usability requirements. Fine-grained citations are necessary to give accurate credit to authors, as the example shows. However, if the granularity is very fine (e.g. every line

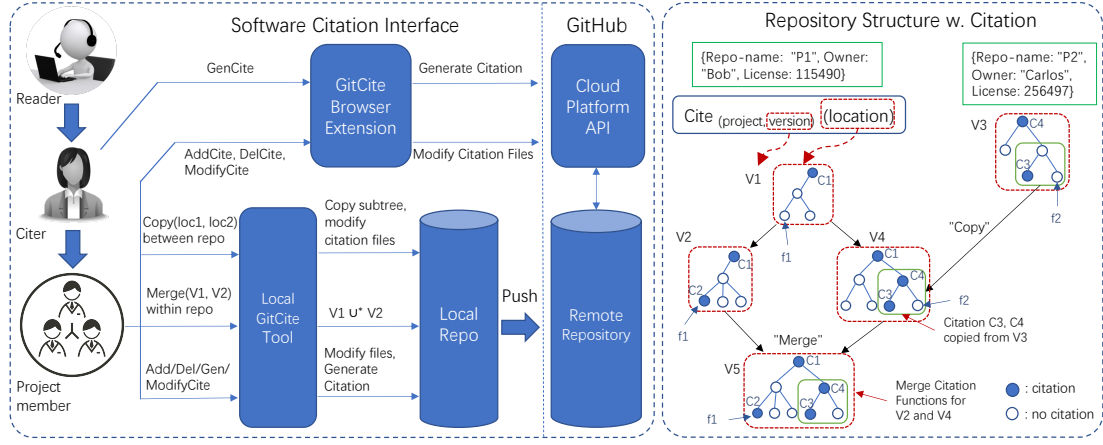


Figure 1. Architecture of GitCite (left) and Running Example (right).

has its own citation), then usability and modularity may be impaired – the workload on the user will increase and little metadata will be shared. In addition to usability issues, a line-level citation model is not compatible with a Git version-control model, and may introduce significant storage overhead. Our solution therefore balances these concerns, and considers citations at the level of files and directories. It also separates citations from code, enabling modularity and usability.

Contributions. This paper presents a model for software citation with version control based on a notion of a *citation function* (Section II). The citation function is designed at the level of files and directories, although it could be extended to the level of subsets of lines. Citations can be specified for a subset of nodes in a directory structure and inherited by other nodes in the directory; the citation is also under version control. We show an interface called GitCite based on our model which is built on top of Git [6] and GitHub [7] (Section III). GitCite allows users to manage citations in their own repositories and “inherit” citations from other repositories. In the demo (Section IV), we will first give an overview of our citation system and introduce the main components. Next, we will present a short video to demonstrate how the citation system is used. Finally, attendees will be able to interact with the system to obtain, create, modify, and delete citations.

Other Related Works. In software engineering community, some *package management* tools like PyPI and Maven have been made and the dependency metadata of the tools is used to track the popularity of repositories [8]. Some other tools like git-subtree and git-submodule [9] focus on managing partially organized source code. These approaches typically aim at setting up software environment while citations [10] focus on claiming authorship and giving credit.

II. CITATION MODEL

We now define the notion of a citation function, describe how it is modified as a result of various operations (add/delete/modify citations, copy/fork/merge subtrees), and discuss how this addresses our technical challenges.

Roles of Users. To clarify the capabilities of users within the citation system, we start by describing the roles of users. Citation involves three types of users: project member, citer,

and reader. A *project member* can edit a project repository, issue Git operations, and manage citations attached to project components (directories or files). A *citer* requests the citation for some component of a project (called GenCite), and may inject the citation into a versioned or unversioned research product (article, paper, website or other code repository). A *reader* sees a citation and may dereference it to obtain the software being cited.

Citation function. A project repository is a directed acyclic graph of project *versions*. Each version is a directory, i.e. a rooted tree whose interior nodes are directories and leaves are files. To address granularity, we allow citations to be defined for any node in any version of a repository. Each version V in project P has an associated *citation function* $\mathcal{C}_{(V,P)}$ which, given a path in the tree, returns a citation for the node at the end of the path. The root of each version *must* be in the active domain, and its citation includes basic snippets of information such as the owner and name of the repository, the http address or DOI of the version, and the version number and/or date.

To address modularity, the citation function may be partial, i.e. some of the directories/files may not have explicit citations attached. Nodes with no explicit citations may inherit the citation from other nodes (defined in a policy). In this demo, we define the citation of a node as follows: Given a node n in version V of project P with root r , the citation of n , denoted $\text{Cite}_{(V,P)}(n)$, is defined as follows:

- $\mathcal{C}_{(V,P)}(n)$, if n has a citation defined (i.e. n is in the active domain of the citation function)
- $\mathcal{C}_{(V,P)}(a)$, where a is the closest ancestor of n for which a citation is defined

Since the root must be in the active domain, $\mathcal{C}_{(V,P)}(n)$ is defined for every node.

Adding/deleting/modifying citations. For usability, it should be possible to update citations to reflect the development of the source code repository. As a project version is modified by inserting/deleting/modifying or renaming files, project members may also modify its citation function by adding (*AddCite*), deleting (*DelCite*), or modifying (*ModifyCite*) citations. Each of these operators takes as input the path of the file/directory whose citation is being modified;

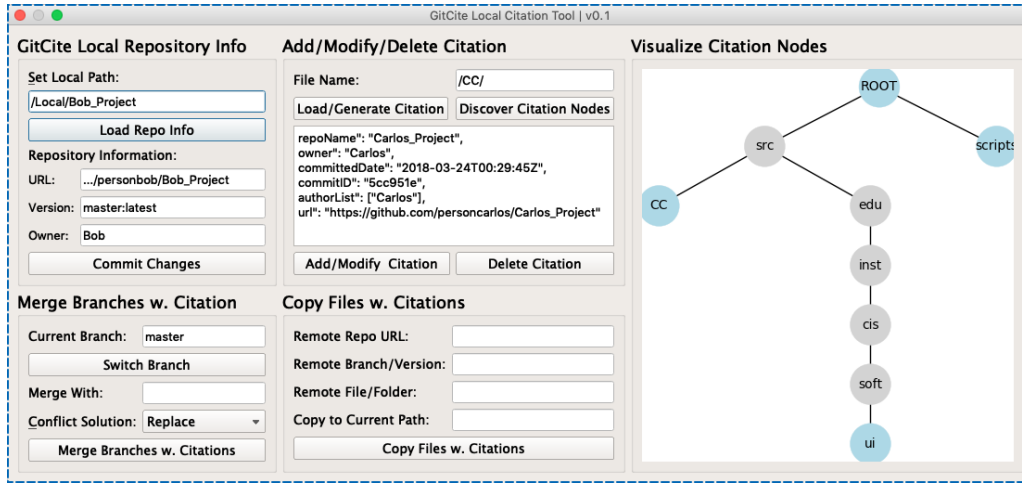


Figure 2. GitCite Local Executable Tool.

AddCite and *ModifyCite* additionally take the value for the new or modified citation. These operations may also be applied by the system as a side-effect of Git operations on the repository. Note that in version control systems such as Git, updates to the project are append only. Therefore, GitCite allows citation updates only on the latest version of a branch.

Carrying citations through Git operators. Another aspect of usability is to keep citations attached to files and directories as they are moved. For example, branches may occur in a repository by separate edits being made to the same version. At some point, two branches $V1$ and $V2$ may be *merged*. A subdirectory T may also be *copied* from version $V1$ of repository $P1$ to version $V2$ of repository $P2$, or a new project $P3$ may be *forked* from the current version of a project $P1$. Through each of these operations, the citation function associated with the new version must be made *consistent* with new directory structure and the files retained in the new version. It should also be as *complete* as possible (modulo conflicts). We will discuss how we implement *MergeCite*, *CopyCite* and *ForkCite* in more detail in the next section.

Example: The right half of Figure 1 illustrates these ideas. Each red-dotted box shows a version of a repository. In particular, $V3$ is a version of project $P2$ while the others are versions of project $P1$. The tree structure in each box shows the directory structure for the repository at a commit point. A solid blue circle means the node (file or directory) has an attached citation and the blue rimmed circle means that there is no attached citation. All versions have a default citation attached to the root. From $V1$ to $V2$, an *AddCite* operation attaches a citation to the leftmost leaf node, $f1$. Therefore, before adding the citation $Cite_{(V1,P1)}(f1)=C1$, whereas afterwards $Cite_{(V2,P1)}(f1)=C2$.

III. ARCHITECTURE

We now describe the architecture of our citation system (see left half of Figure 1). CitCite consists of two components: a *browser extension*, and a *local executable tool*. Together they provide a citation service for GitHub software repositories. Since the local executable tool is based on Git, it is also

compatible with any other online project management website which uses Git. We discuss how citation functions are stored before describing the components.

Storing Citation Functions. To address modularity and usability, we store citations in a special citation file at the root of each version of a Git repository. The file is a set of key-value entries, where the key is the relative path to the file being cited, and the value is the citation attached to the file. It cannot be directly modified by users, but is updated as a side-effect of the various citation and Git operations. When a new commit version of the project is created, the updated citation file is also created which reflects the updates made since the previous version. These updates include adding/deleting/modifying entries as a result of *AddCite*, *DelCite*, or *ModifyCite* operations; changing the key of an entry to reflect a renamed file or directory; or adding new entries in response to copying/merging between branches. Thus, much of the work involved in storing and maintaining citations is done as a side-effect of versioning in Git.

Browser extension. The GitCite browser extension is deployed on Chrome, and can generate citations for remote repositories on GitHub. The extension communicates with the GitHub server using its API, and directly modifies the citation file on the remote repository to reflect changes to the citation function. The popup page shown in Figure 3 illustrates its functionality: If the user is a citer, clicking on “Generate Citation” will immediately generate the citation based on the citation file in that version; recall that this is either the citation explicitly attached to the node or that of its closest ancestor. The returned citation can then be copy-pasted to their local bibliography manager. Since citers are not allowed to add/delete/modify files, they will not be allowed to use the “Add/Delete” button functionalities. However, if the user is a project member, the text box will display the citation explicitly attached to the node, if it exists, which the user may then modify. If an explicit citation does not exist, the user may either enter a citation, or use the “Generate” button to see the citation of its closest ancestor, which can then be modified for the current node. If no citation has ever been enabled for the

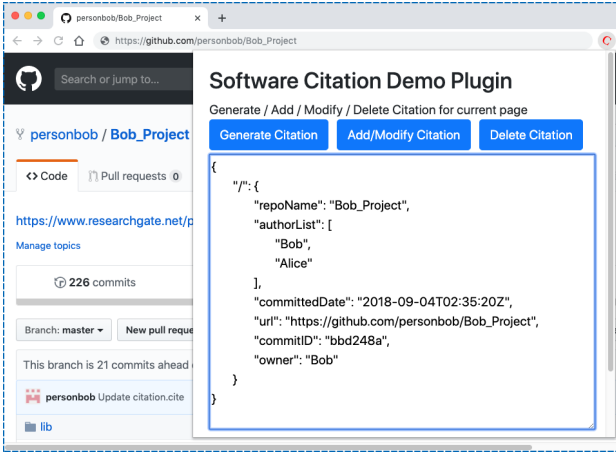


Figure 3. GitCite Browser Extension.

repository (as is the case for most existing repositories), then a default citation would be created from the metadata of the GitHub repository, and the project member could use it as a draft that they could further edit before saving.

Local executable tool. When a project member downloads a copy of the project repository with Git, the GitCite local executable tool (Figure 2) can be used to manage the citation file in the download. In addition to *AddCite*, *DelCite*, and *ModifyCite*, it also implements the *CopyCite*, *MergeCite* and *ForkCite* functions. *CopyCite* copies a directory from a remote repository version to the local repository version, and *migrates* their associated citations. That is, the citations for the directory and its subtree in the remote citation file are added to the local citation file, with the key paths modified to reflect the new location to ensure correctness of the citation function. *MergeCite* merges two branches in the same repository, and merges the citation files while resolving conflicts. Although Git conflict resolution rules are used for all regular files, we do not use them on the citation file since it could leave the citation function inconsistent. Instead, we simply take the union of the citation files, and delete any entries that correspond to files that were deleted by the Git merge. Conflicts over the values associated with the same key in the new citation file are then resolved by showing them to the user and asking the user to resolve the conflict. More complex conflict resolution strategies could also be used. *ForkCite* copies a version of a repository, along with its history, and creates a new repository. The citations in the citation file are also copied. Our way of storing citations will naturally enable *ForkCite* through GitHub’s *Fork*. Users can also view the nodes with explicit citations attached in light blue on the right side of the tool. When changes to files and their citations are finally committed, the Git push command is used to push the local copy (which contains the citation file) to the remote repository.

Example, cont. Returning to the right half of Figure 1, when the subtree of $V3$ in the green box is copied from project $P2$ to $P1$ by *CopyCite*, the attached citations are migrated as described earlier. Since $f2$ does not have an associated citation in $V3$, before the copy event $Cite_{(V3,P2)}(f2)=C4$, which is the citation of its closest ancestor. Note that $C4$

is also the citation for the root of the subtree being copied. After copying to $P1$ the citation for $f2$ remains unchanged: $Cite_{(V4,P1)}(f2)=C4$ since the citation for the root of the green subtree from $V3$ has been added to the citation file for $V4$, indicated by the root now being a solid blue color. Finally, $V2$ and $V4$ are merged using *MergeCite* to produce $V5$, in which the citations are also merged along with the files. In this example there are no conflicts, so we simply take the union of the citation files.

IV. DEMONSTRATION SCENARIO

During the demo, we will talk about the need for fine-grained software citation, explain our citation model, and show the GitCite architecture. We will then demonstrate both the browser extension and the local executable functionalities of GitCite, showing how citations are managed through changes as well as how they can be added or modified within the current repository. Using the example in the introduction, we will illustrate how our tools can be used to establish citations for the whole repository, such that different citations will be returned for different components. After this introduction, attendees will be invited to test our citation tools using the demo machine, either assuming the role of a citer to obtain citations from repositories of their choosing, or assuming the role of a project member to generate/modify citations, or to copy/merge citations among our sample project repositories.

V. CONCLUSION

This demo presents a model and implementation called GitCite for citing software project repositories managed by version control systems like Git. The system consists of a browser extension that works with GitHub, and a local executable tool that works with any project management platform using Git. Citations may be explicitly attached to any subset of the nodes to show its identity and authorship, and the root of the project must always have a default citation. The citation for a node is based on its *closest ancestor* with an explicit citation. Users are able to create, manage and use citations through *AddCite*, *DelCite*, and *ModifyCite* functions, as well as the citation-extended Git functions *CopyCite*, *MergeCite* and *ForkCite*.

REFERENCES

- [1] Panel Summary of Reproducibility in ML Workshop, ICML’17. <https://sites.google.com/view/icml-reproducibility-workshop/icml2017>.
- [2] M. Martone, Ed., *Data Citation Synthesis Group: Joint Declaration of Data Citation Principles*. FORCE11, 2014.
- [3] A. Smith, D. Katz, K. Niemeyer, and FORCE11 Software Citation Group, “Software citation principles,” *PeerJ Computer Science*, 2016.
- [4] Zenodo. <https://zenodo.org>.
- [5] The Mozilla Science Team. (2016) FORCE11 Software Citation Tools. <https://github.com/mozillascience/software-citation-tools>.
- [6] L. Torvalds and J. Hamano. (2010) Git. <http://git-scm.com>.
- [7] GitHub. <https://github.com/>.
- [8] Listing the packages that a repository depends on. <https://help.github.com/en/github/visualizing-repository-data-with-graphs/listing-the-packages-that-a-repository-depends-on>.
- [9] Git Tools - Subtree Merging. <https://git-scm.com/book/en/v1/Git-Tools-Subtree-Merging>.
- [10] A. Alawini, L. Chen, S. B. Davidson, N. P. D. Silva, and G. Silvello, “Automating data citation: The eagle-i experience,” in *JCDL*, 2017.