Dear Dr. Pottegard,

We would like to thank you for your response and the invitation to resubmit a revised version of our manuscript "The FAIRification of research in real-world evidence: A practical introduction to reproducible analytic workflows using Git and R" submitted to Pharmacoepidemiology & Drug Safety (Manuscript ID: PDS-23-0292).

We are thankful for the constructive and thoughtful comments provided by the reviewers in evaluating this manuscript. We acknowledge the valuable insights and have refined the manuscript to align with the reviewers' recommendations. Please see our detailed responses one by one below.

We believe that these revisions have significantly enhanced the quality and contribution of our work which we feel is now well-suited for publication in Pharmacoepidemiology & Drug Safety.

Thank you for your consideration of our revised manuscript.

Sincerely,

Janick Weberpals, RPh, PhD

Reviewers' Comments to Author:

# Reviewing: 1

Comments to the Author

The manuscript is a brief introduction and an effective tutorial for a set of tools aimed at assist traceability and reproducibility of the programming code executing a RWE study.

The text is very clear and well-written, and is a timely and useful document. I add few suggestions, that the author may feel free to disregard.

[RESPONSE] We thank the reviewer for the interest in this manuscript and the very thoughtful comments which clearly improved the quality of the manuscript.

============

Introduction

============

"guidance on transparency regarding the actual implementation and analytic steps is still significantly lacking, especially when analyses are implemented via line coding in statistical programming languages"

Is line coding discussed here as opposed to interface-driven programming? The latter would hardly seem more transparent/reproducible. The sentence could be perhaps clarified.

[RESPONSE] We thank the reviewer for pointing this out and agree that reproducible and transparent processes are necessary and applicable to both line coding as well as interface-driven analyses of data. We hence rephrased the sentence to be more general and agnostic of the way the analysis is implemented.

**Section 1. Introduction**: “*While many advances have been made in the documentation and reporting of study protocols and results5–8, guidance on transparency regarding the actual implementation and analytic steps is still significantly lacking*.”

"By design, VCS workflows automatically empower users to comment, track and compare changes made to files and hence, increase the ability to comprehend the evolution of a project over time while maintaining a complete audit trail of changes to all documents"

A similar concept is mentioned in the discussion. I invite the authors to expand slightly on this recommendation, and consider whether an ethical dimension should be encompassed. For a long time, transparency in the process that underlies the evolution of protocols and reports has been recommended, as a means to ensure that decisions are made based on scientific arguments. This is considered particularly important when the study is sponsored by an organization that has financial interests, or other types of interests, in the outcome of the study. For example, the ENCePP Code of Conduct recommends that a brief description of the contribution to the protocol of all parties involved is stored in the public website EU PAS, and similarly for reports. I wonder whether the authors think that the same motivation applies to transparency in the evolution of programming code. Otherwise, what is the motivation underlying this recommendation? Purely in terms of reproducibility, access the final version of the programming code would seem to be sufficient.

[RESPONSE] We thank the reviewer for this insightful comment and the opportunity to further elaborate on the recommendation regarding version control systems (VCS) workflows and their role in the transparency of code evolution. We agree that the access to the final version of the programming code is in most cases already sufficient for the reproducibility among external researchers. However, the advantages of having a complete audit trail of all changes made to a studies’ codebase (similar to study protocol amendments) are two-fold. First, for an internal study team it is important to be able to track changes over time to comprehend any potential errors that may have been introduced in previous steps and which may be hard to trace back otherwise. Second, for both internal and external researchers the complete audit trail of the code not only facilitates reproducibility but also aids in understanding the decision-making process, identifying the scientific rationale behind specific changes, understanding quality control processes (e.g., code review) and tracking the contributions of different collaborators to the codebase.

We added these aspects to the discussion:

**Section 4. Discussion**: *“While the access to the final version of the programming code is already sufficient for external reproducibility, the tracking of changes over time enables a study team to comprehend and resolve errors more efficiently. Similar to tracking changes and amendments of a study protocol, an external audience also benefits from a complete audit trail through a better understanding of the documented scientific rationale behind specific changes, the identification of implemented quality control processes (e.g., code reviews) and the tracking of contributions of different collaborators to the codebase.”*

============================================================

A step-by-step introduction on how to use Git in a RWE study

============================================================

The manuscript correctly puts the use of Git and GitHub in the context of the FAIR principles. A cornerstone of FAIR is the use of persistent identifiers. It may be useful to mention and discuss this concept, and point the readers to the means to create persistent identifiers associated to GitHub objects, see for instance ([LINK](https://secure-web.cisco.com/1CHgrFfR6PSg3yBk0B2bdzB6KqqaSh2ToulfIOtTztSMV5IdfwsEP2V4C1rHrWU3roxjbzZl0VWCGykG5OZlnclcX5853CboJugjL4o0izUeEpho3YS7p7U-_iVEhYjVxSESud-_SG5z70AdqH8mWLdspjJReckJ9vqas2Mv1mM1J3Sy3OoPMRpJsuqsQN3M3uacRuXY4EX7NW7WxNk3GKHPaf0Lm_D1Dy-ovNAhfS0jUAOzxPXJyIKsbIGFrmAvvY5UJVjeP6dj_92vi1VTuRlaiMbst59IrUFIDLPFh2UapQu9_pFnsYcUA-eTY2A4W/https%3A%2F%2Fdocs.github.com%2Fen%2Frepositories%2Farchiving-a-github-repository%2Freferencing-and-citing-content))

[RESPONSE] The reviewer addresses an important aspect of the FAIR principles in terms of persistent identifiers to identify code repositories. We agree with the reviewer that this is not only a crucial aspect of transparency and reproducibility in general, but another benefit of remote repositories as code can be easily shared and uniquely referenced. We added sub-section 3.8 that discusses the role of persistent identifiers and how to create such for GitHub code releases utilizing the Zenodo platform. We also created a digital object identifier for the repository that accompanies this publication and added it to the corresponding README file on GitHub and GitLab.

**Section 3.8 Creating persistent identifiers**: *“A cornerstone of FAIR principles is the use of persistent identifiers, such as digital object identifiers (DOI), for enabling findability and accessibility of research objects. Integrating the concept of persistent identifiers within the context of remote repositories is an important aspect to enhance the traceability and accessibility of code repositories or even specific commits or releases of a study codebase.*

*To that end, Zenodo, a free and open-source platform funded by the European Commission and developed and maintained by European Organization for Nuclear Research (CERN), allows users to share diverse research artifacts. The platform seamlessly interfaces with GitHub, enabling the creation of referenceable DOIs for existing code repositories. The steps required to obtain a DOI involve the linkage of a Zenodo and a GitHub account. Once linked, a user will be able to select a specific release of a repository which automatically registers a new DOI and a corresponding badge which can be included and referenced, e.g., in a GitHub README file.”*

===========

Discussion

===========

The authors may consider indicating that to achieve full replicability, it is recommended to make available in the repository a sample of dummy data, formatted in such a way that the program code would succesfully run.

[RESPONSE] The reviewer makes a great point and we added this aspects to the discussion:

**Section 4.Discussion**: *“In order to achieve full replicability, authors may alternatively include a sample of dummy data to enable readers to comprehend and run the analytic code. Moreover, many larger research data sources have already provided synthetic data using identical formats and variable names while protecting patient privacy (e.g., Medicare Claims Synthetic Public Use Files).”*

The common data models that the authors mention were all originated in the US. They may consider adding to the list the ConcePTION CDM, which was designed to accomodate data source of more heterogeneous origin.

[RESPONSE] We thank the reviewer for making us aware of the ConcePTION CDM which we added to the manuscript text.

================================

Typos

page 7: 'neccessary'

[RESPONSE] Typo was corrected in the manuscript text.

Reviewing: 2

Comments to the Author

This is about very important tools primarily implemented as version control systems in pharmacoepidemiology. In that sense, some procedures and git commands explained in the paper are just a small example of all that can be done. Even though some important commands are missing, like merge, rebase, tag, and fetch. Fetch is the most basic remote command from Git and needs to be explained. This are all basic commands, as the paper is not presenting the advanced commands.

[RESPONSE] We thank the reviewer for highlighting the merge, rebase, tag, and fetch commands which had not been featured yet. Also addressing the below reviewer comment on the order of git command in a workflow ("always pull before push"), we now present the fetch, merge, rebase and pull commands as the first step of the git workflow in the sub-section 3.5.1 and updated Figure 3 accordingly.

**Section 3.5.1 Synchronize changes from a remote repository to a local repository (pull, fetch, merge and rebase)**: *“Especially when working collaboratively with multiple users on a remote repository, it is usually good practice to synchronize the status of one's local repository and the remote repository before making any changes since there could be potential modifications and updates already made by others. To avoid any conflicts between two different versions and always work with the most recent version of a file, the first step of the git workflow involves a git pull or git fetch command.*

* *git fetch downloads any changes made to a remote repository into a user's local repository. However, the command isolates the potentially modified files from the state of the existing local files. This is useful since it enables users to safely review changes using git log (we will cover this in more detail in Section 3.7) before integrating those locally. To eventually allow the integration of these changes locally, users can do so using the git merge or git rebase command. The difference between these two commands is how the history of the versions between different snapshots of the repository is tracked, for which git merge is more common, simpler and less error-prone and git rebase is a more advanced feature (for the purpose of this manuscript we will mostly focus on git merge).*

*[CODE EXAMPLE]*

* *git pull combines git fetch and git merge in one command and directly downloads and integrates all changes from a remote repository locally.*

*[CODE EXAMPLE]*

*Occasionally it can occur that two collaborators made conflicting changes to the same file or even to the same line of code. While Git is generally good at automatically integrating (merging) new changes, Git will notify the user to resolve a major conflict by editing the conflicting file if Git cannot automatically determine what is correct (e.g., if one collaborator has made changes to a line of code and the other completely deleted it). […]”*

We also introduce the git tag command separately within a dedicated “Tags” sub-section.

**Section 3.5.3 Tags (tag):** *“A handy Git feature to highlight particularly important commits, snapshots, code/software releases or milestones of a project is tagging. In context of RWE studies, this can be a specific version of an analysis, e.g., the one used for a regulatory submission or the completion of a manuscript version or revision.*

*Tags can be annotated with a semantic identifier and tag message, e.g.,*

*[CODE EXAMPLE]*

*All created tags can be displayed using the following command:*

*[CODE EXAMPLE]”*

Git is a powerful "time machine" where users can see the old code and compare it to recent versions. Perhaps this is the most important feature of Git and it is not explained.

[RESPONSE] We fully agree with the reviewer that the tracking of versions/snapshots of a repository over time is potentially the most important Git feature overall. We have tried to explain emphasize this aspect already in the previous version of the manuscript and now added a few sentences to highlight this even more at several points in the manuscript:

* **Section 2. Git in a nutshell**: “[…] *The way to conceptually think about how Git works is that it makes a snapshot of a research project repository every time the state of a project is saved and hence can be seen as a "time machine" that enables researchers and developers to compare code and repositories across different versions over time*.”
* **Section 3.5.2. Stage and commit changes**: *“Git allows researchers to keep track of changes by making snapshots of the repository every time the state of a project is saved. In Git terms, this fundamental step is referred to as a commit and it enables users to compare code versions across snapshots. Generally, it is up to the user when to commit, which files to commit and how the commit should be documented in a commit message. However, it is best practice to commit early and often. That is, smaller and more granular commits that reflect a single unit of work (e.g., a change of selected confounders to adjust for) make it easier to comprehend the changes and associated analytic results and revert potential errors.”*
* **Section 3.7. Tracking changes**: *“[…] One of the main motivations to adhere to this workflow is that in return we get rewarded with a detailed audit trail of changes made to our study files.”*
* **Section 4. Discussion**: *”* *Despite the potential initial complexities to get acquainted with the Git workflow, distributed VCS and code sharing provide unique advantages, such as a full audit trail of which changes were made by whom, when and why. […] While the access to the final version of the programming code is already sufficient for external reproducibility, the tracking of changes over time enables a study team to comprehend and resolve errors more efficiently. Similar to tracking changes and amendments of a study protocol, an external audience also benefits from a complete audit trail through a better understanding of the documented scientific rationale behind specific changes, the identification of implemented quality control processes (e.g., code reviews) and the tracking of contributions of different collaborators to the codebase”*

There is no need to explain Renv when you are presenting Git. Renv is another topic (computational environment isolation), and it's not closely related to VCS. This brings confusion and lack of clarity in the goals.

[RESPONSE] We removed the section on Renv/computational environments in R from the main manuscript. However, we consider this aspect of computational reproducibility as important and complementary to Git and hence moved the section to supplementary material and reference this in the discussion:

**Section 4. Discussion**: “*Besides the usage of Git and remote repositories as a version control systems, there are also further relevant aspects that determine the reproducibility of analytic code when using open-source software such as R. To complement the introduction and discussion on Git in this manuscript, section 2 (Reproducible computation environments in R) in the Supplementary Material provides a few further considerations addressing the computational reproducibility when using R*.”

The "Absolute versus relative paths session" is not related to Git. Consider removing this section.

[RESPONSE] We removed the section on "Absolute versus relative paths session" from the main manuscript. However, we consider this aspect as important and complementary to Git and hence moved the section to supplementary material and reference this in the discussion:

**Section 4. Discussion**: “*Besides the usage of Git and remote repositories as a version control systems, there are also further relevant aspects that determine the reproducibility of analytic code when using open-source software such as R. To complement the introduction and discussion on Git in this manuscript, section 2 (Reproducible computation environments in R) in the Supplementary Material provides a few further considerations addressing the computational reproducibility when using R*.”

The paper introduces some topics in the wrong order. It is traditional when teaching Git that you need first to Pull, and then Push. The paper introduces the commands in the inverted order. In general, the paper does not introduce best practices, and commands are presented like they are in the Git manual. Consider to include what are the best practices using Git, like "always pull before push".

[RESPONSE] We agree with the reviewer regarding the best practice to “always pull before push”. We updated the order accordingly and present the introduction of git fetch, git merge, git rebase and git pull as the first occurrence in a git workflow (see response to the first comment of reviewer #2 above). This now also reflected in the updated Figure 3.

RStudio was rebranded to POSIT. Why do you still refer to it by the old name? This makes the content look dated.

[RESPONSE] We agree with the reviewer that the RStudio company was rebranded to POSIT, however in our manuscript we refer to the RStudio integrated development environment (IDE) which is still called “RStudio” (see screenshot below). We adjusted our wording in the manuscript to explicitly refer to the RStudio IDE:

**Section 3. A step-by-step introduction on how to use Git in a RWE study**: “*In this section, we want to give a technical introduction on how Git and remote repositories can be used in RWE studies. For the following examples, we focus on workflows utilizing* ***the RStudio integrated development environment (IDE) GUI*** *(https://posit.co/download/rstudio-desktop/) and GitHub (https://github.com/) as a remote repository, although the general concepts are fairly similar and easily transferable to other programming languages and remote repository platforms, respectively*.”

A screenshot of a website

Description automatically generated

Security is a mandatory topic when scientists use Git and Github, Gitlab, etc, it's not explained how POSIT is integrated into these. I am missing explanations on how to grant access, authorizations, passwords, how to use teams, discussions, private vs open repositories, and Personal access tokens.

[RESPONSE] We acknowledge the reviewer’s comment regarding security and added a paragraph about security considerations when working with Git, remote repositories and discuss how sensitive information such as personal access tokens or passwords can be managed in the RStudio IDE using environment variables and how to configure *a .gitignore* file so that these kind of data are not tracked and accidentally shared. Given that this is an introduction, we think that with this paragraph we cover the most important security aspects.

**Section 3.9 Security considerations**: “*When working with Git and remote repositories, several security considerations are crucial to safeguard sensitive information. First, it is of utmost importance to ensure that sensitive information (such as personal access tokens, passwords or patient-level data) are not accidentally shared. One way to prevent this is to configure a .gitignore file, which is a flat text file typically stored in the root directory of the repository. Each line in .gitignore specifies a file name, sub-directory name or a file name pattern (e.g., "\*.csv" for all files of type csv) that a user intentionally does not want to track and synchronize with a remote repository. If it can be avoided, such sensitive information should alternatively not be stored in the project directory in the first place.*

*[EXAMPLE]*

*Second, it is good practice to not include passwords or personal access tokens inside an analytic code file but to call these from an external file. RStudio, for example, offers the possibility to create environment files (.Renviron and .RProfile) which are user-controllable text files which can be stored in the project root directory and is automatically called upon an R session start. Such a file can be configured to not be tracked using .gitignore (as illustrated above) and accessed as show in the following.*

*[EXAMPLE]*

*These credentials can now be called inside an R script, for example, to access a database (example taken from genieBPC R package26).*

*[EXAMPLE]*

*Third, it needs to be ensured that access permissions to remote repositories are appropriately configured, granting only necessary access levels to collaborators unless it is intended to share the repository with the public. All major remote repositories offer the option to configure a repository to be private or public.*

”

Missing about importance to have instructions in Readme.rd and how to set a license, this is especially important for open-source software.

[RESPONSE] We thank the reviewer for highlighting this important aspect of open-source projects. We added a comprehensive paragraph to the sub-chapter *“Repository structure, README file and licenses”*, highlighting the importance of providing all necessary information about the repository in the README.md file and further provide some guidance on how to choose the right license for a given open-source software.

***Section 3.4 Repository structure, README file and licenses****: “All relevant metadata, instructions and information about a project should be documented in the README.md file. This file can be auto-generated when a project is initialized through a remote repository (see prior step in Section 3.3.2) or can be manually added afterwards by creating a text file with the name README.md in the root directory of the repository. Essential information to include contain a brief summary about the project’s background and objectives, instructions on how to install software or manage computation environments and dependencies, important files and the organization of the repository, contribution guidelines, contact information and the project’s license to outline the terms under which others can use, modify, and distribute the project. Especially for open-source software projects choosing a suitable license is critical for which GitHub provides helpful guidance and information under https://choosealicense.com/.”*

Professional companies and even other scientists are promoting videos and courses that are going much deeper and provide a better introduction to this topic. The article is important because it focuses on how Git may be used in our own field, and this is a must in pharmacoepidemiology, so I would recommend focusing on what is necessary to start using Git, how to merge, solve conflicts, compare code in different points in time and how to release final versions of the open-source code to the community.

[RESPONSE] We thank the reviewer for the valuable and thoughtful feedback on our manuscript draft. We are convinced that after having addressed the reviewer’s comments above strengthened the article's quality and focus on the essentials of Git within the field of pharmacoepidemiology. We acknowledge in the discussion that this is an introductory article and compiled a comprehensive list in the supplementary material with further readings, materials, videos and courses which interested readers can use to deepen their knowledge on Git.

**Section 4. Discussion**: “[…] *This tutorial is by no means comprehensive as Git provides a vast number of additional features which were not addressed in this manuscript. To get familiar with more advanced Git usage, we compiled a non-exhaustive list of further suggested readings and resources in the Supplementary Material.”*

Reviewing: 3

Comments to the Author

Dear colleagues,

I am glad to see an introduction to the use of GIT and transparent practices for R for the RWE. This is a very good summary of some key aspects one should know when developing an analytical script as part of RWE/pharmaco-epidemiology. In this manuscript, you go through Version Control, Code Structure, Dependency management, Virtual environment, and Use of Containers, which are most of the practices and techniques for reproducible programming. However, I am missing one key element, Documentation in the code as well as the use of other documentation files such as codebooks. Code documentation is an essential practice for reproducible code. In order to have a fully complete manuscript, I strongly suggest including this as a subsection in the Manuscript.

[RESPONSE] We thank the reviewer for the insightful comments and for highlighting the importance of code documentation in the context of reproducible programming. We acknowledge the significance of thorough documentation, including code comments and supplementary documentation files like codebooks, as essential practices for ensuring reproducibility in analytical scripts within RWE and pharmaco-epidemiology. Along with another reviewer comment that touched on the documentation of a given study repository, we now emphasize the importance of codebooks, data dictionaries and other essential metadata of a RWE study repository in section 3.4.

***Section 3.4 Repository structure, README file and licenses****: “All relevant metadata, instructions and information about a project should be documented in the README.md file. This file can be auto-generated when a project is initialized through a remote repository (see prior step in Section 3.3.2) or can be manually added afterwards by creating a text file with the name README.md in the root directory of the repository. Essential information to include contain a brief summary about the project’s background and objectives, instructions on how to install software or manage computation environments and dependencies, important files and the organization of the repository, contribution guidelines, contact information and the project’s license to outline the terms under which others can use, modify, and distribute the project. Especially for open-source software projects choosing a suitable license is critical for which GitHub provides helpful guidance and information under* [*https://choosealicense.com/*](https://choosealicense.com/)*.*

***A special aspect about real-world data, such as administrative insurance claims and electronic health records, is that these data are typically not collected for the primary purposes of doing research but rather for reasons of billing or clinical documentation. Hence, being transparent about the data provenance, pre-processing steps and operationalization of exposures, outcomes and covariates is of critical importance20 and should be part of the repository’s metadata. This can come in form of codebooks, data dictionaries, literate programming scripts (e.g., Quarto) or other applicable documentation files. Developing reproducible and well documented functions or packages as well as inline commenting of programming code can further enhance the transparency and readability of analytic code.21”***

*20. Wang SV, Schneeweiss S, Berger ML, et al. Reporting to Improve Reproducibility and Facilitate Validity Assessment for Healthcare Database Studies V1.0. Pharmacoepidemiology and Drug Safety 2017; 26: 1018–1032. doi:10.1002/pds.4295.*

*21. Bové DS, Seibold H, Boulesteix A-L, et al. Improving software engineering in biostatistics: Challenges and opportunities. 2023. Available at: https://arxiv.org/abs/2301.11791*

Also, I would like to see more details regarding the specific challenges that an RWE research team might face when using Git.

[RESPONSE] We agree with the reviewer that there are also challenges associated with the usage of Git and added a paragraph on how this paper/Git is relevant for diverse members of RWE teams (see reviewer comment below) and associated challenges.

***Section 4. Discussion:*** *“Git was primarily designed for version control and development of analytic code by statisticians and programmers, but may also benefit other members of a RWE generation team through the centralized deployment, tracking and version control of study documents as well as project management tools which have become standard among established remote repository platforms. While Git offers substantial benefits for collaboration, reproducibility, and transparency in RWE teams, challenges can arise in its adoption, particularly related to merging conflicts, steep learning curves, and maintaining consistent usage practices across team members.”*

The RWE team is very diverse, from epidemiologists and doctors to statisticians and data managers. I would strongly suggest making a remark on how this paper is relevant for each of the members of an RWE generation team.

[RESPONSE] See response to reviewer comment above.

Finally, I have minor comments that I believe should be added to bring clarity on the manuscript. These are:

- Page 7, Lines 16 and 17. As described further down in the text. One of the key characteristics of Git, including the versioning controls, is that Git can describe changes in files well when the contents can be rendered as plain text. So, generally, people understand the files listed as examples in line 16 and 17 of page 7 as excel, word, PNG, and PDF files. These extensions do not allow for tracking changes, so this needs to be clarified in the manuscript.

[RESPONSE] We fully agree with the reviewer that this is a limitation that should be highlighted in the manuscript, which we do in the fourth paragraph of the discussion:

***Section 4. Discussion:*** *“Git was primarily designed to work with source code files and works best for plain, text-based documents such as programming code, rich text format (RTF), Markdown or LaTex files. There are limitations when it comes to tracking binary files like Word (.docx) and Excel (.xlsx) as these need to be interpreted by a program or a hardware processor. While it can be still useful to track versions of these file types using Git, it is important to recognize that modifications in such documents can’t be easily inspected using RStudio or GitHub.”*

- Figure 2. Related to the previous point. In Figure 2, you show a folder structure of an RWE project which includes .qmd files (which are not common file types) and are used for the protocol and manuscript. This should be explained in the caption.

[RESPONSE] We added an explanation for this file type in Figure 2:

*“ […] \*.qmd = Markdown-specific format based on the Quarto open-source scientific and technical publishing system.”*

- Page 10, Line 41. As well as describing the folder structure. In the readme.md file, the license should be stated. This is a key characteristic that could affect the FAIRness of a project. You could add a short explanation of the different licenses and guidance on how to select one in the supplementary appendix.

[RESPONSE] We thank the reviewer for highlighting this. This was also raised by another reviewer and we now stress the need to include a license and refer to [*https://choosealicense.com/*](https://choosealicense.com/) *which is a fantastic resource to determine an appropriate license type for a given repository.*

*“Essential information to include contain a brief summary about the project’s background and objectives, instructions on how to install software or manage computation environments and dependencies, important files and the organization of the repository, contribution guidelines, contact information and* ***the project’s license to outline the terms under which others can use, modify, and distribute the project. Especially for open-source software projects choosing a suitable license is critical for which GitHub provides helpful guidance and information under https://choosealicense.com/.”***

- Page 30, line 24. You use the term ‘main’ to refer to the principal development branch. However, in the figure below, you are using Master. Moreover, the master is not a term used anymore ([LINK](https://secure-web.cisco.com/1t3ZXucNSImFmTeGR9stP6-Phx4En56fEEDmYzRYTJXY-bpzmU7SYxdZr_lsTsbnmGu-PFfwAjVtfhqKaWLI3Yvi_KbGf8MHqXSldpCFHfwk9V_IiPHI9AUnvFgc5CbFu2n37tQKqbURu_2azE0rkUT_I9nouYTdXQT3AnbidcCKHhVl1MEuwPLChBEj7MtVg3aIUFFdYwDaSvEUqgOuEzkdxkfxVoATq9--enaCKVpnC3Sak-U7HRz2pGM_4EnzkiTdBbIrGOi3J-MDP9Zd_PR2IpGBIF7oR_K-5O3oD93OkikoJrWMP93QrVh3PnOQO/https%3A%2F%2Fwww.theserverside.com%2Ffeature%2FWhy-GitHub-renamed-its-master-branch-to-main)). Please update or rephrase in the manuscript.

[RESPONSE] We rephrased this accordingly.

- Page 31, You are switching between the terms main and master, and it’s confusing. Please adjust accordingly in the manuscript.

[RESPONSE] We rephrased this accordingly.

- Figure 6 I couldn’t find where the box ‘Commit SHA token’ is pointing to. Please adjust accordingly.

[RESPONSE] The box ‘Commit SHA token’ points to the abbreviated SHA token for this specific commit: “commit 9b1a94c”. We added a short explanation of this to the figure caption:

***Figure 6****: “[…] The unique abbreviated SHA token for this specific change/commit is '9b1a94c' (see 'Commit SHA token' box on the upper right of the figure).”*