The FAIRification of research in real-world evidence: A practical introduction to reproducible analytic workflows using Git and R

FAIRification of research in real-world evidence

Janick Weberpals[1,*],
Shirley V. Wang[1]

[1]Division of Pharmacoepidemiology and Pharmacoeconomics, Department of Medicine, Brigham and Women'~~'~~s Hospital~~,~~
Harvard Medical School~~,~~
Boston, Massachusetts~~MA~~, USA

[*]~~Correspondence:~~ Janick Weberpals, ~~RPh, PhD,~~ Division of Pharmacoepidemiology and Pharmacoeconomics, Department of Medicine, Brigham and Women'~~'~~s Hospital, Harvard Medical School, 1620 Tremont Street, Suite 3030-R, Boston, MA 02120, USA.
~~, Phone: 617-278-0932, Fax: 617-232-8602, Email:~~ jweberpals@bwh.harvard.edu

Data S1. Supporting ~~lementary material~~Information: Supplementary Appendix and ~~Supplementary~~ Figures (pdf).

~~FUNDING INFORMATION~~~~Funding~~

~~There was no specific funding for this manuscript.~~

~~Disclosures/COI~~

~~Janick Weberpals and Shirley Wang report no conflicts of interest.~~

~~Analytical code sharing statement: This manuscript was written using Quarto version 1.3.433 (https://quarto.org/) and R version 4.1.2. All materials can be found at https://gitlab-scm.partners.org/drugepi/fair-epi or https://github.com/janickweberpals/fair-epi. Detailed information on packages and versions can be found in the renv.lock file in the linked repository.~~

ACKNOWLEDGMENTS~~Acknowledgments~~

~~None.~~

Abstract

Transparency and reproducibility are major prerequisites for conducting meaningful real-world evidence (RWE) studies that are fit for decision-making. Many advances have been made in the documentation and reporting of study protocols and results, but the principles for version control and sharing of analytic code in RWE are not yet as established as in other quantitative disciplines like computational biology and health informatics. In this practical tutorial, we aim to give an introduction to distributed version control systems (VCS) tailored toward~~s~~ the FAIR (Findable, Accessible, Interoperable, and Reproducible) implementation of RWE studies. To ease adoption, we provide detailed step-by-step instructions with practical examples on how the Git VCS and R programming language can be implemented into RWE study workflows to facilitate reproducible analyz~~s~~es. We further discuss and showcase how these tools can be used to track changes, collaborate, disseminate, and archive RWE studies through dedicated project repositories that maintain a complete audit trail of all relevant study documents.

KEYWORDS

Git

R

r~~R~~eal-world evidence

r~~R~~eproducibility

t~~T~~ransparency

v~~V~~ersion control

Key Points~~KEY POINTS~~

- Transparency and reproducibility are major prerequisites for conducting meaningful real-world evidence (RWE) studies that are fit for decision-making.
- Regulators, funding agencies, and HTA bodies have increasing expectations regarding the provenance, audit trial, and sharing of study documents including analytic code as a prerequisite for impactful and credible RWE studies.
- Version control systems (VCS) for analytic code like Git, GitHub, and GitLab have unique advantages for making analytic code and other relevant RWE study documents Findable, Accessible, Interoperable, and Reproducible (FAIR).
- In this manuscript, we discuss and demonstrate how VCS can be used to track changes over the course of a RWE project, collaborate, disseminate, and archive analytic code and provide a technical step-by-step tutorial for the practical implementation of reproducible workflows using Git.
- Given the trend toward~~s~~ ~~open~~ open-source tools for clinical trial reporting and regulatory submissions, this tutorial addresses a timely topic and aims to encourage new and experienced pharmacoepidemiologists to integrate VCS in their daily work and embrace the advantages of FAIR analytic code sharing.

1. INTRODUCTION

Real-world evidence (RWE) studies that make secondary use of routinely collected health data captured in electronic health records and claims are increasingly being used to inform drug development, regulatory, and coverage decisions, as well as clinical practice.[1–4] Considering the impact that RWE studies can have on patient healthcare, transparent, and reproducible conduct of RWE studies is critically important.

While many advances have been made in the documentation and reporting of study protocols and results,[5–8]~~,~~ guidance on transparency regarding the actual implementation and analytic steps is still significantly lacking. Although in other quantitative disciplines, such as computational biology[9–12] or health informatics,[13]~~,~~ the sharing of code is rather the rule than the exception, this practice is not as common in the (pharmaco-) epidemiological community. As ~~open~~ open-source tools like R packages and Shiny dashboards[14] become increasingly used for clinical trial reporting and regulatory submissions,[15]~~,~~ the adoption of version control and code sharing practices is a crucial and timely topic for the field of RWE.

Version control systems (VCS), such as *Git*, provide powerful tools to keep track of the versioning of important files and documents, such as protocols, analytic code, tables, and figures. Thereby, VCS can extend the principles of *FAIR data*[16] to the implementation of pharmacoepidemiological study workflows by making all components of a study Findable, Accessible, Interoperable, and Reproducible (FAIR).

VCS enables researchers to track and resolve errors, collaborate with peers, and share resources instantaneously, for example, upon publication of a manuscript. 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.[17] Without VCS, this is often naturally done by assigning uninformative file names, such as “*final_analysis_1_rev_more_changes.R*," ~~”~~ which is bad practice as this is highly ~~error~~ error-prone, lacking transparency for collaborators, and changes are difficult to track and reconcile. Particularly in disciplines like pharmacoepidemiology, where analy~~s~~zes can have ~~far reaching~~far-reaching impact on healthcare policy and practice decisions, analytic code needs to be accessible and reproducible.

To increase the adoption of transparent and reproducible workflows in RWE studies using VCS, this tutorial aims to give a practical introduction for pharmacoepidemiologists on how to set up, structure, and implement workflows using *Git*, which is the most popular VCS to date.[18] We will first give a brief introduction to Git and its usage for collaboration and dissemination of study results through project repositories. Then we will provide a technical step-by-step guidance on how to integrate Git in analytic RWE workflows. We will additionally discuss and showcase important aspects of reproducibility using the R open-source programming language, although the basic principles are applicable to any major coding language.

2. GIT IN A NUTSHELL

Git is a free and ~~open~~ open-source distributed VCS software, which was developed in 2005 by the LINUX developer community primarily with the intent to handle large software projects efficiently.[19] 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. For our purposes, a repository can be thought of as an isolated project directory, which includes all necessary files and documents for a given research study (e.g., protocol, programming code, manuscript, tables, and figures, etc.).

In its basic form, Git is a local software and does not need any internet connectivity or connection to a remote server to store *local* changes made to files in a repository. However, to leverage the full potential of Git, it is almost always used in combination with web-based remote repository hosting services, such as *GitHub*, *GitLab*, ~~*Bitbucket*~~ *Bitbucket*, or other platforms (henceforth referred to as remote repositories, Overview Box 1).

BOX 1 Differences between Git and remote repository platforms (GitHub, GitLab, etc.)

Git

- Is an open-source software and distributed ~~version control system~~VCS.
- Generally operated through command line tools, graphical user interfaces are available.
- Installed locally on a system to track local changes of a Git repository.

Remote repository platforms (GitHub, GitLab, etc.)

- Web-based services, usually provided by a commercial entity.
- Usually free for basic functions, paid services for more advanced features.
- Needed to collaborate on shared repositories and dissemination of code.
- Offer a graphical user interface and other functionalities that complement basic Git functions.

These remote platforms make it possible to synchronize local Git repositories across multiple project members and thereby improve collaboration. They further complement Git functionalities by providing a graphical user interface (GUI) to visualize changes made to files and often offer advanced features, such as the automation of workflows using continuous integration and deployment (CI/CD) or, most recently, AI-powered virtual coding assistants (e.g., GitHub co-pilot). Generally speaking, it i~~'~~s possible to use Git without a remote repository, while it i~~'~~s unusual to use a remote repository without Git.

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. We will show examples for both command line interface (CLI) prompts as well as for RStudio'~~’~~s integrated GUI. For more details, we refer to the book *Pro Git* by Chacon and Straub.[19]

3.1. Installing Git

Git is available for every contemporary operating system and can be downloaded on the official Git website: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

To check if the installation succeeded, the following prompt in the CLI can be used, which will output the available version.

```
# check installed git version
git --version
```

3.2. Configuration and first-time use

For Git to be able to track who made changes, the first step is to provide a name and email address as shown below. This needs to be done only once if the --global parameter is set.

```
# global configuration of name and email address
git config --global user.name "First Last"
git config --global user.email "name@domain.edu"
```

3.3. Initialization of a new Git-controlled repository

3.3.1. Local initialization

To start a new project (e.g., a new RWE study), the next step is to create and navigate to a new directory where all documents that are intended to be tracked via Git will be stored (in the following referred to as *repository*). In this example, the study repository is called rwe_study.

```
# create and navigate to study directory
mkdir ~/rwe_study
cd ~/rwe_study
```

The command to initialize this directory as a Git repository is as follows:~~.~~

```
# initialize git tracking
git init
```

This initiates the creation of the .git sub~~-~~directory within the study repository, which will contain the entire version history of the rwe_study project. Typically, users do not~~don''t~~ directly interact with this sub~~-~~directory and the only thing to keep in mind is that this folder keeps all files needed to track changes.

3.3.2. GUI initialization and git clone

An alternative to creating a new repository locally is through initiating a new project in the remote repository (in this case GitHub) and then creating a local copy, a process which is usually referred to as cloning.

To do so, users will need to log into their remote repository account and navigate to the section New repository (~~Supplementary~~ Figure S1). This will take a user to a form to populate basic information of the new project, such as the repository name (rwe_study) and choose basic settings, such as if the repository should be private or public and if README.md (see Section 3.4) and .gitignore (see Section 3.5.2 and Section 3.9) files should be automatically created (~~Supplementary~~ Figure S2).

Now we can copy the pre~~-~~populated repository to our local machine by copy~~-~~pasting the link given under Clone > Copy HTTPS into the CLI (~~Supplementary~~ Figure S3):

```
# create and navigate to study directory and clone
# the git repository which was already created on GitHub
mkdir ~/rwe_study
cd ~/rwe_study
git clone https://github.com/USER/rwe_study.git
```

This can also be achieved directly in the RStudio GUI via File > New Project > Version Control > Git > paste the link to Repository URL (Figure 1).

In case a user wants to join or contribute to an already existing project, git clone works the same way with the only difference that the repository already contains some files and is not empty.

3.4. Repository structure, README file and licenses

At this step, we now can populate the repository and start working on study-relevant files and analyz~~s~~es. To enhance clarity, it is recommended to use a logical structure and an informative nomenclature for sub~~-~~directories and file names (a minimal example is illustrated in Figure 2).

All relevant metadata, ~~instructions~~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~~projects, choosing a suitable license is critical for which GitHub provides helpful guidance and information under 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~~preprocessing steps and operationalization of exposures, outcomes, and covariates is of critical importance[20] 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]

To speed up these initial steps, we provide a pre~~-~~populated template of such a structured study repository based on the HARPER protocol[8] template under https://gitlab-scm.partners.org/drugepi/harper. HARPER is a harmonized study protocol template endorsed by the International Society of Pharmacoepidemiology and the International Society of Pharmacoeconomics and Outcomes Research. The pre~~-~~populated structured study repository template is publicly available and can be forked[22] to initiate a new project (see instructions in the template repository under the provided link).

3.5. The Git workflow

The most essential steps of a git workflow are summarized in Figure 3 and in the Supporting Information~~Supplementary Material~~.

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).

```
# fetch ("download") updated version of a remote repository
git fetch

# use merge to integrate these changes into local files using merge
git merge
```

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

```
# download and integrate content from remote to local
git pull
```

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).

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~~a single unit of work (e.g., a change of selected confounders to adjust for) makes it easier to comprehend the changes and associated analytic results and revert potential errors.

To determine which files should be selected (or staged) for a new commit, the git add command is used followed by the name(s) of the files that should be committed or a “~~’~~.~~'~~” to stage all changes present in the current repository.

```
# Examplary staging command
git add 'scripts/03_propensity_score_analysis.R'
```

Once all files that should be committed in the same step are staged, the git commit command will create a local snapshot of the changes. The -m suffix and quoted text after the command represent the commit message, ~~i.e.~~that is, an informative yet brief comment on what changes were performed.

```
# Examplary commit command
git commit -m 'added title to plot illustrating propensity score overlap'
```

Every time a user makes a commit to create a snapshot of the work, Git creates a unique hash (a 40-character string created by a simple hashing algorithm [SHA]) which can be seen as an “ID” of the snapshot which enables users to comprehend every change made and revert back to any snapshot of the repository. Figure 4 illustrates the equivalent steps in the RStudio GUI.

If any file(s) in the study repository should not be tracked and hence not be synchronized with the remote repository (e.g., because they include confidential information that should not be visible to others like database credentials), these can be specified in the .gitignore file which is typically located in the root directory of the repository. To create a .gitignore file, a user can either auto-generate it upon initialization of the repository on GitHub (~~Supplementary~~ Figure S2) or manually add a text file called .gitignore to the root directory (more information in Section 3.9).

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.~~for example, 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.~~for example,

```
git tag -a v2.0 -m "Revised (R1) manuscript version"
```

All created tags can be displayed using the following command:

```
it tag
```

3.6. Synchronize changes with remote repository

3.6.1. Synchronize local changes to remote (push)

To synchronize the local changes, the next step involves the upload of our commits to the remote repository on GitHub via the push command.

```
# push local changes to remote repository
git push
```

In case we initialized the local repository as described in ~~section~~ Section 3.3.2, a remote was automatically configured. This can be checked via the following command.

```
# check remote configuration
git remote -v
```

If there is no remote established yet, this can be retrospectively configured using the following commands.

```
# add a new remote repository connection
git remote add origin https://github.com/USER/rwe_study.git
git branch -M main
git push -u origin main
```

Tags, as described in Section 3.5.3, need to be separately pushed, ~~e.g.~~for example,

```
git push origin v2.0
```

In summary, git push is not needed if the intention of a researcher is just to keep track of local changes and there is no wish to share code, collaborate with others or work across different systems. However, most of the times researchers use Git particularly for the purpose of collaborating on a project as depicted in Figure 5.

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. Probably the best way to visualize and track changes is by browsing the commit history on the remote repository. Figure 6 illustrates this for the example commit we have previously performed in Section 3.5.2. A similar view for uncommitted changes in the RStudio GUI is also provided in Figure 4.

For completeness, tracking uncommitted changes is also possible via the command line, but usually provides the user with a less intuitive output compared to what is visually provided by RStudio and GitHub.

```
# display differences between uncommitted changes
git diff
```

A complete summary of committed changes with details on the SHA hash, author, email, time, and commit messages can be accessed via~~.~~ the following command line:

```
# display commit history
git log
```

While there are many more useful Git features, such as branching (see ~~Supplementary Material~~Supporting Information), the workflow of staging, committing, and pushing/pulling (Figure 3) may already suffice for the majority of pharmacoepidemiologists.

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,[23, 24]~~,~~ 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.[25]~~.~~ 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 in a GitHub README file.

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 .gitgnore file specyfing which
# files, sub-directories and patterns
# that should not be tracked

# password .txt file
pwd.txt

# Session Data files
.RData
.Rhistory

# any type of .csv file
*.csv

# output directory
output/

# .Renviron file
.Renviron
```

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: configuration of database access credentials inside an .Renviron file
db_username = "JoshSmith"
db_password = "User1234"
```

These credentials can now be called inside an R script, for example, to access a database (example taken from genieBPC R package[26]).

```
# Example how the credentials defined in .Renviron can be used
# inside an R script
set_synapse_credentials(
  username = Sys.getenv("db_username"),
  password = Sys.getenv("db_password")
  )
```

Third, it needs to be ensured that access permissions to remote repository platforms 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.

4. DISCUSSION

In this practical tutorial, we introduced basic concepts of VCS for transparent and reproducible workflows and gave technical step-by-step instructions for the implementation using Git and R. We discussed frequently occurring challenges for reproducibility and how these can be overcome.

Git is a powerful tool that perfectly integrates into workflows of quantitative disciplines handling analytic code. It further provides features that make it convenient for project-oriented setups and collaboration at scale. 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~~Supporting Information.

Besides the usage of Git and remote repositories as 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 Supporting Information~~**Supplementary Material**~~ provides a few further considerations addressing the computational reproducibility when using R.

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.[18] 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 cannot~~can't~~ be easily inspected using RStudio or GitHub.

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. Remote repositories are ideal platforms to deploy and archive scripts and documents, facilitate collaboration and make it very convenient to share code. While the access to the final version of the programming code along with the source data is 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.

Although data sharing is unfortunately often prohibited due to privacy and legal reasons, many databases have been mapped to common data models like OMOP,[27]~~;~~ Sentinel,[28]~~;~~ PCORnet,[29] and ConcePTION,[30]~~;~~ which makes shared code interoperable even without direct access to the same data source. 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).[31] Authors may also include a sample of dummy data to give readers a better understanding of the underlying source code and enhance transparency, but, by definition, reproducibility can only be achieved by having access to real data.

Finally, code sharing can enable researchers to learn from each other, build trust,[32]~~;~~ expedite the uptake of novel methodologies and knowledge exchange and decrease unnecessary redundancies through sharing of code libraries, functions and algorithms. 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.

Regulators,[33]~~;~~ funding agencies,[34, 35] and HTA bodies[36] have increasing expectations regarding the provenance, sharing and audit trial of study documents including analytic code as a prerequisite for impactful and credible RWE studies. Given the trend toward~~s~~ ~~open~~ open-source tools for clinical trial reporting and regulatory submissions, this tutorial addresses a timely topic and hopefully encourages new and experienced pharmacoepidemiologists to integrate VCS in their daily work and embrace the advantages of FAIR analytic code sharing.

FUNDING INFORMATION

There was no specific funding for this manuscript.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflict of interest.

DATA AVAILABILITY STATEMENT

This manuscript was written using Quarto version 1.3.433 (https://quarto.org/) and R version 4.1.2. All materials can be found at https://gitlab-scm.partners.org/drugepi/fair-epi or https://github.com/janickweberpals/fair-epi. Detailed information on packages and versions can be found in the renv.lock file in the linked repository.

REFERENCES~~References~~

1. United States Food and Drug Administration. Framework for FDA'~~'~~s real world evidence program. 2018. Accessed ~~6~~June~~/~~ 30~~/~~, 2023. ~~Available at:~~https://www.fda.gov/downloads/ScienceResearch/SpecialTopics/RealWorldEvidence/UCM627769.pdf.

2. Eichler H-G, Baird L, Barker R~~,~~, Bloechl-Daum B., Børlum-Kristensen F., Brown J., Chua R., del Signore S., Dugan U., Ferguson J., Garner S., Goettsch W., Haigh J., Honig P., Hoos A., Huckle P., Kondo T., le Cam Y., Leufkens H., Lim R., Longson C., Lumpkin M., Maraganore J., O'Rourke B., Oye K., Pezalla E., Pignatti F., Raine J., Rasi G., Salmonson T., Samaha D., Schneeweiss S., Siviero P.D., Skinner M., Teagarden J.R., Tominaga T., Trusheim M.R., Tunis S., Unger T.F., Vamvakas S., Hirsch G.~~et al.~~ From adaptive licensing to adaptive pathways: d~~D~~elivering a flexible life-span approach to bring new drugs to patients. Clin Pharmacol Ther~~Clinical Pharmacology & Therapeutics~~ 2015; 97: 234–246. doi:10.1002/cpt.59.

3. CIOMS. CIOMS RWE and RWD for regulatory decision making. 2023. ~~Accessed 6/30/2023~~Accessed June 30, 2023. ~~Available at:~~https://cioms.ch/wp-content/uploads/2020/03/CIOMS-WG-XIII_6June2023_Draft-report-for-comment-1.pdfA.

4. Makady A, ten Ham R, de Boer A, Hillege H, Klungel O, Goettsch W. Policies for u~~U~~se of r~~R~~eal-w~~W~~orld d~~D~~ata in h~~H~~ealth t~~T~~echnology a~~A~~ssessment (HTA): a~~A~~

cC~~C~~omparative s~~S~~tudy of s~~S~~ix HTA a~~A~~gencies. Value Health~~Value in Health~~ 2017; 20: 520–532. doi:10.1016/j.jval.2016.12.003.

5. Wang SV, Pinheiro S, Hua W~~,~~, Arlett P, Uyama Y, Berlin JA, Bartels DB, Kahler KH, Bessette LG, Schneeweiss S~~et al.~~ STaRT-RWE: structured template for planning and reporting on the implementation of real world evidence studies. BMJ 2021: m4856. doi:10.1136/bmj.m4856, 372.

6. Schneeweiss S, Rassen JA, Brown JS~~,~~, Rothman KJ, Happe L, Arlett P, Dal Pan G, Goettsch W, Murk W, Wang SV~~et al.~~ Graphical d~~D~~epiction of l~~L~~ongitudinal s~~S~~tudy d~~D~~esigns in h~~H~~ealth c~~C~~are d~~D~~atabases. Ann Intern Med~~Annals of Internal Medicine~~ 2019; 170: 398. doi:10.7326/m18-3079, 406.

7. Berger ML, Sox H, Willke RJ~~,~~, Brixner DL, Eichler HG, Goettsch W, Madigan D, Makady A, Schneeweiss S, Tarricone R, Wang SV, Watkins J, Daniel Mullins C~~et al.~~ Good practices for real-world data studies of treatment and/or comparative effectiveness: r~~R~~ecommendations from the joint ISPOR-ISPE s~~S~~pecial t~~T~~ask f~~F~~orce on real-world evidence in health care decision making. Pharmacoepidemiol Drug Saf~~Pharmacoepidemiology and Drug Safety~~ 2017; 26: 1033–1039. doi:10.1002/pds.4297.

8. Wang SV, Pottegård A, Crown W~~,~~, Arlett P, Ashcroft DM, Benchimol EI, Berger ML, Crane G, Goettsch W, Hua W, Kabadi S, Kern DM, Kurz X, Langan S, Nonaka T, Orsini L, Perez-Gutthann S, Pinheiro S, Pratt N, Schneeweiss S, Toussi M, Williams RJ~~et al.~~ HARmonized p~~P~~rotocol t~~T~~emplate to e~~E~~nhance r~~R~~eproducibility of hypothesis evaluating real-world evidence studies on treatment effects: a~~A~~ good practices report of a joint ISPE/ISPOR task force. Pharmacoepidemiol Drug Saf~~Pharmacoepidemiology and Drug Safety~~ 2022; 32: 44–55. doi:10.1002/pds.5507.

9. Almugbel R, Hung L-H, Hu J~~,~~, Almutairy A, Ortogero N, Tamta Y, Yeung KY~~et al.~~ Reproducible ~~B~~bioconductor workflows using browser-based interactive

notebooks and containers. J Am Med Inform Assoc~~Journal of the American Medical Informatics Association~~ 2017; 25: 4–12. doi:10.1093/jamia/ocx120.

10. Mammoliti A, Smirnov P, Nakano M~~,~~, Safikhani Z, Eeles C, Seo H, Nair SK, Mer AS, Smith I, Ho C, Beri G, Kusko R, Massive Analysis Quality Control (MAQC) Society Board of Directors, Lin E, Yu Y, Martin S, Hafner M, Haibe-Kains B~~*et al.*~~ Orchestrating and sharing large multimodal data for transparent and reproducible research ~~Nature~~ Nat Commun~~ications~~ 2021; ~~12~~5797. doi:10.1038/s41467-021-25974-w, 12.

11. Russell PH, Johnson RL, Ananthan S, Harnke B, Carlson NE. A large-scale analysis of bioinformatics code on GitHub. ~~Qin Z (ed.).~~ PLoS One~~PLOS ONE~~ 2018; 13: e0205898. doi:10.1371/journal.pone.0205898.

12. Perez-Riverol Y, Gatto L, Wang R, *et al.* Ten s~~S~~imple r~~R~~ules for t~~T~~aking a~~A~~dvantage of Git and GitHub. ~~Markel S (ed.).~~ PLoS Comput Biol~~PLOS Computational Biology~~ 2016; 12: e1004947. doi:10.1371/journal.pcbi.1004947.

13. Bakken S. The journey to transparency, reproducibility, and replicability. J Am Med Inform Assoc~~Journal of the American Medical Informatics Association~~ 2019; 26: 185–187. doi:10.1093/jamia/ocz007.

14. Pharmaverse. Accessed June~~6/~~ 30~~/~~, 2023. ~~Available at:~~ https://pharmaverse.org/.

15. Kephart C. R packages and shiny for FDA clinical trial submissions. Accessed June 30, 2023~~Accessed 6/30/2023~~. ~~Available at:~~ https://posit.co/blog/fda-shiny-r-package-submissions/.

16. García-Closas M, Ahearn TU, Gaudet MM~~,~~, Hurson AN, Balasubramanian JB, Choudhury PP, Gerlanc NM, Patel B, Russ D, Abubakar M, Freedman ND, Wong WSW, Chanock SJ, Berrington de Gonzalez A, Almeida JS~~*et al.*~~ Moving t~~T~~oward f~~F~~indable, a~~A~~ccessible, i~~I~~nteroperable, r~~R~~eusable p~~P~~ractices in e~~E~~pidemiologic r~~R~~esearch. Am J Epidemiol~~American Journal of Epidemiology~~ 2023; 192: 995–1005. doi:10.1093/aje/kwad040.

17. Ram K. Git can facilitate greater reproducibility and increased transparency in science. Source Code Biol Med~~Source Code for Biology and Medicine~~ 2013; 7. doi:10.1186/1751-0473-8-7, 8.

18. Blischak JD, Davenport ER, Wilson G. A q~~Q~~uick i~~I~~ntroduction to v~~V~~ersion c~~C~~ontrol with Git and GitHub. ~~Ouellette F (ed.).~~ PLoS Comput Biol~~PLOS Computational Biology~~ 2016; 12: e1004668. doi:10.1371/journal.pcbi.1004668.

19. Chacon S, Straub B. Pro Git. Springer Nature, 2014.

20. Wang SV, Schneeweiss S, Berger ML~~,~~, Brown J, de Vries F, Douglas I, Gagne JJ, Gini R, Klungel O, Mullins CD, Nguyen MD, Rassen JA, Smeeth L, Sturkenboom M, on behalf of the joint ISPE-ISPOR Special Task Force on Real World Evidence in Health Care Decision Making~~et al.~~ Reporting to i~~I~~mprove r~~R~~eproducibility and f~~F~~acilitate v~~V~~alidity a~~A~~ssessment for h~~H~~ealthcare d~~D~~atabase s~~S~~tudies V1.0. Pharmacoepidemiol Drug Saf~~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.

22. Fork a repo. A fork is a new repository that shares code and visibility settings with the original "upstream" repository. Accessed ~~6~~June~~/~~ 30~~/~~, 2023. ~~Available at:~~ https://docs.github.com/en/get-started/quickstart/fork-a-repo.

23. European Commission, Directorate General for Research and Innovation., Lisbon Council., CWTS., ESADE., Elsevier. ~~Directorate General for Research and Innovation., Lisbon Council., CWTS., ESADE., Elsevier.~~ Zenodo: O~~o~~pen S~~s~~cience M~~m~~onitor C~~c~~ase S~~s~~tudy. LU: Publications Office, 2019. doi:10.2777/298228.

24. Zenodo. Zenodo. Accessed November 15, 2023. ~~Available at:~~ https://zenodo.org.

25. Zenodo. Zenodo. Accessed November 15, 2023. ~~Available at:~~ https://zenodo.org/account/settings/github/.

26. Lavery JA, Brown S, Curry MA, Martin A, Sjoberg DD, Whiting K. A data processing pipeline for the AACR project GENIE biopharma collaborative data with the genieBPC R package. ~~Wren J (ed.).~~ Bioinformatics 2022; btac796~~39~~. doi:10.1093/bioinformatics/btac796 39.

27. OHDSI~~.~~ The book of OHDSI: ~~O~~observational health data sciences and informatics. ~~OHDSI,~~ 2019. ~~Available at:~~https://ohdsi.github.io/TheBookOfOhdsi/.

28. Brown JS, Maro JC, Nguyen M, Ball R. Using and improving distributed data networks to generate actionable evidence: the case of real-world outcomes in the Food and Drug Administration'~~'~~s s~~S~~entinel system. J Am Med Inform Assoc~~Journal of the American Medical Informatics Association~~ 2020; 27: 793–797. doi:10.1093/jamia/ocaa028.

29. Fleurence RL, Curtis LH, Califf RM, Platt R, Selby JV, Brown JS. Launching PCORnet, a national patient-centered clinical research network. J Am Med Inform Assoc~~Journal of the American Medical Informatics Association~~ 2014; 21: 578–582. doi:10.1136/amiajnl-2014-002747.

30. Thurin NH, Pajouheshnia R, Roberto G~~, ~~, Dodd C, Hyeraci G, Bartolini C, Paoletti O, Nordeng H, Wallach-Kildemoes H, Ehrenstein V, Dudukina E, MacDonald T, de Paoli G, Loane M, Damase-Michel C, Beau AB, Droz-Perroteau C, Lassalle R, Bergman J, Swart K, Schink T, Cavero-Carbonell C, Barrachina-Bonet L, Gomez-Lumbreras A, Giner-Soriano M, Aragón M, Neville AJ, Puccini A, Pierini A, Ientile V, Trifirò G, Rissmann A, Leinonen MK, Martikainen V, Jordan S, Thayer D, Scanlon I, Georgiou ME, Cunnington M, Swertz M, Sturkenboom M, Gini R~~*et al.*~~ From i~~I~~nception to ConcePTION: g~~G~~enesis of a n~~N~~etwork to s~~S~~upport b~~B~~etter m~~M~~onitoring and c~~C~~ommunication of m~~M~~edication s~~S~~afety d~~D~~uring p~~P~~regnancy and b~~B~~reastfeeding. Clin~~ical~~ Pharmacol~~ogy &~~ Ther~~apeutics~~ 2021; 111: 321–331. doi:10.1002/cpt.2476.

31. Gonzales A, Guruswamy G, Smith SR. Synthetic data in health care: a~~A~~ narrative review. ~~Johnson A (ed.).~~ PLOS Digital Health 2023; 2: e0000082. doi:10.1371/journal.pdig.0000082.

32. Orsini LS, Monz B, Mullins CD~~,~~, van Brunt D, Daniel G, Eichler HG, Graff J, Guerino J, Berger M, Lederer NM, Jonsson P, Schneeweiss S, Wang SV, Crown W, Goettsch W, Willke RJ~~et al.~~ Improving transparency to build trust in real-world secondary data studies for hypothesis testing— ~~– W~~Why, what, and how: recommendations and a road map from the real-world evidence transparency initiative. Pharmacoepidemiol Drug Saf~~Pharmacoepidemiology and Drug Safety~~ 2020; 29: 1504–1513. doi:10.1002/pds.5079.

33. United States Food and Drug Administration. Considerations for the use of real-world data and real-world evidence to support regulatory decision-making for drug and biological products (draft guidance). 2021. Accessed ~~7~~July~~/~~ ~~0~~3~~/~~, 2023. ~~Available at:~~ https://www.fda.gov/regulatory-information/search-fda-guidance-documents/considerations-use-real-world-data-and-real-world-evidence-support-regulatory-decision-making-drug.

34. National Institute of Health (NIH). NIH data management plan policy. ~~Jan~~ 2023. Accessed July 3, 2023~~Accessed 7/03/2023~~. ~~Available at:~~https://sharing.nih.gov/data-management-and-sharing-policy/planning-and-budgeting-for-data-management-and-sharing/writing-a-data-management-and-sharing-plan#after.

35. Patient-Centered Outcomes Research Institute (PCORI). Policy for data management and data sharing. Accessed July 3, 2023~~Accessed 7/03/2023. Available at:~~ https://www.pcori.org/about/governance/policy-data-management-and-data-sharing.

36. Kent S, Salcher-Konrad M, Boccia S~~,~~, Bouvy JC, Waure C, Espin J, Facey K, Nguyen M, Rejon-Parrilla JC, Jonsson P~~et al.~~ The use of nonrandomized evidence to estimate treatment effects in health technology assessment. J~~ournal of Comparative~~ Comp ~~Effectiveness~~ Eff Res~~earch~~ 2021; 10: 1035–1043. doi:10.2217/cer-2021-0108.

FIGURE 1: Steps to clone a remote repository using the RStudio integrated development environment (IDE) graphical user interface (GUI).

FIGURE 2: Minimal example of a transparent repository structure containing relevant study documents. ~~Abbreviations:~~ .git = Git sub~~-~~directory which includes all files to keep track of changes (created as a result of git init command), renv = R environment sub~~-~~directory which contains the project library and other project-specific files and settings needed to manage R package versions and the R environment, renv.lock = R environment lockfile, describing the R version and R package versions used in a project, *.qmd = Markdown-specific format based on the Quarto open-source scientific and technical publishing system.

FIGURE 3: Overview of a basic Git workflow.

FIGURE 4: The RStudio integrated development environment (IDE) provides a graphical user interface (GUI) as an alternative to command line prompts to interact with Git and to perform command equivalent to git add, git commit, and git push/pull (red circles). This view can be accessed in the right upper pane of the RStudio IDE under Git -> Diff. In this example, an R script with the title “~~‘~~03_propensity_score_analysis.R”~~’~~ is selected and staged. All changes performed in this file can be viewed in the window below with green highlighted lines indicating

additions and red highlighted lines deletions from the prior version. In this example, a title was added to a propensity score plot which is also expressed in the commit message.

FIGURE 5: Git in connection with remote repositories (e.g., GitHub or GitLab) can signficantly improve collaboration across project members.

FIGURE 6: Remote repositories like GitHub provide tools to visually track changes made to analytical code and corresponding output, such as figures. This commit history illustrates details about the commit (e.g., commit SHA [simple hasing algorithm] and message), a side-by-side comparison of the previous (upper left) and modified (upper right) version of the figure and the corresponding changes made to the R code (bottom; line 32 in the code script was added). The unique abbreviated SHA token for this specific change/commit is “~~‘~~9b1a94c”~~’~~ (see “~~‘~~Commit SHA token”~~’~~ box on the upper right of the figure).