ENCORE: A practical implementation to improve reproducibility and transparency of computational research

Supplementary Information

ENCORE

Step by Step ENCORE Guide

ENhancing COmputational REproducibility

Date: 30 July 2024 **Version:** 13.3	**Prof. dr. A.H.C. van Kampen** Bioinformatics Laboratory Epidemiology and Data Science Amsterdam UMC Amsterdam, the Netherlands a.h.vankampen@amsterdamumc.nl https://www.bioinformaticslaboratory.eu

Contents

1 INTRODUCTION

This *Step-by-Step ENCORE Guide* is part of the Enhancing Computational Reproducibility (ENCORE) initiative of the Bioinformatics Laboratory of the Amsterdam UMC. ENCORE focuses on reproducibility, implying the reanalysis of the same data using the same methods. ENCORE doesn't consider replicability (sometimes referred to as repeatability) which is about strengthening scientific evidence from replication studies by other research groups using independent data, and experimental and computational methods.

Each research or support project that follows the **ENCORE principles** comprises three main components: the **standardized File System Structure (sFSS)** template including a set of pre-defined files, a **GitHub repository**, and the **Compendium Navigator**. This document provides the general philosophy behind the ENCORE principles and a recipe to start a new project according to these principles. The documentation found in this guide complements the specific instructions that are found inside the sFSS template.

Further information about ENCORE is also found in the ENCORE wiki (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/wiki). To facilitate further discussion about the develop of ENCORE we setup Discussions (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/discussions).

Also have a look at **ENCORE AUTOMATION**) for scripts that automate specific aspects of ENCORE (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE_AUTOMATION).

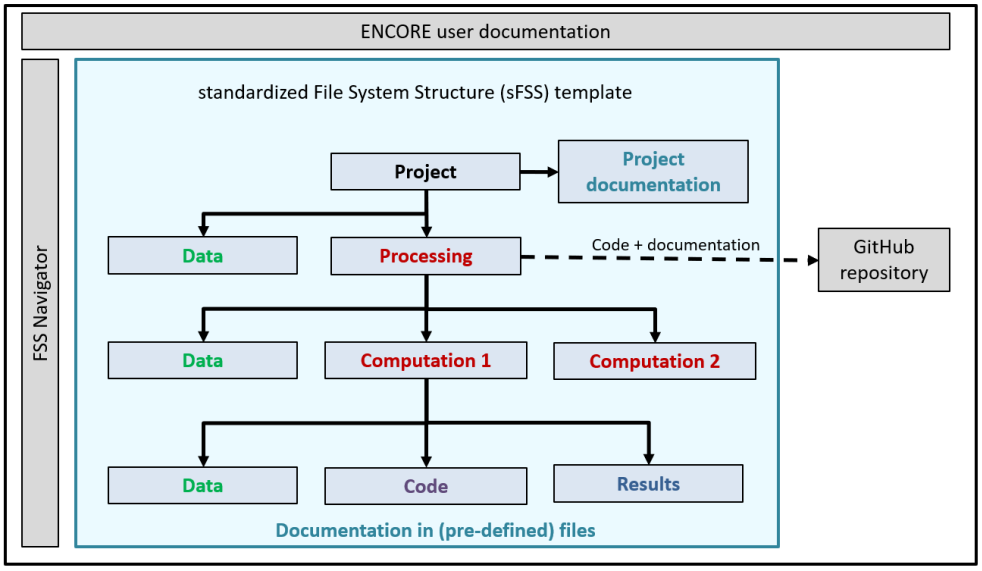
1.1 ENCORE COMPONENTS

ENCORE comprises three components:

1. **The standardized File System Structure (sFSS) template**: This provides a standardized directory structure (template) according to which a project should be organized. It provides sections for the background information, data, and computations. This is also referred to as the **Project Compendium**. The main parts of the template are shown in *Supplementary Figure 1*.

2. **The GitHub repository:** Git and GitHub provide a system for software version control, hosting, and sharing (*Section 6; Appendix*)**.** The repository only contains the software and software documentation.

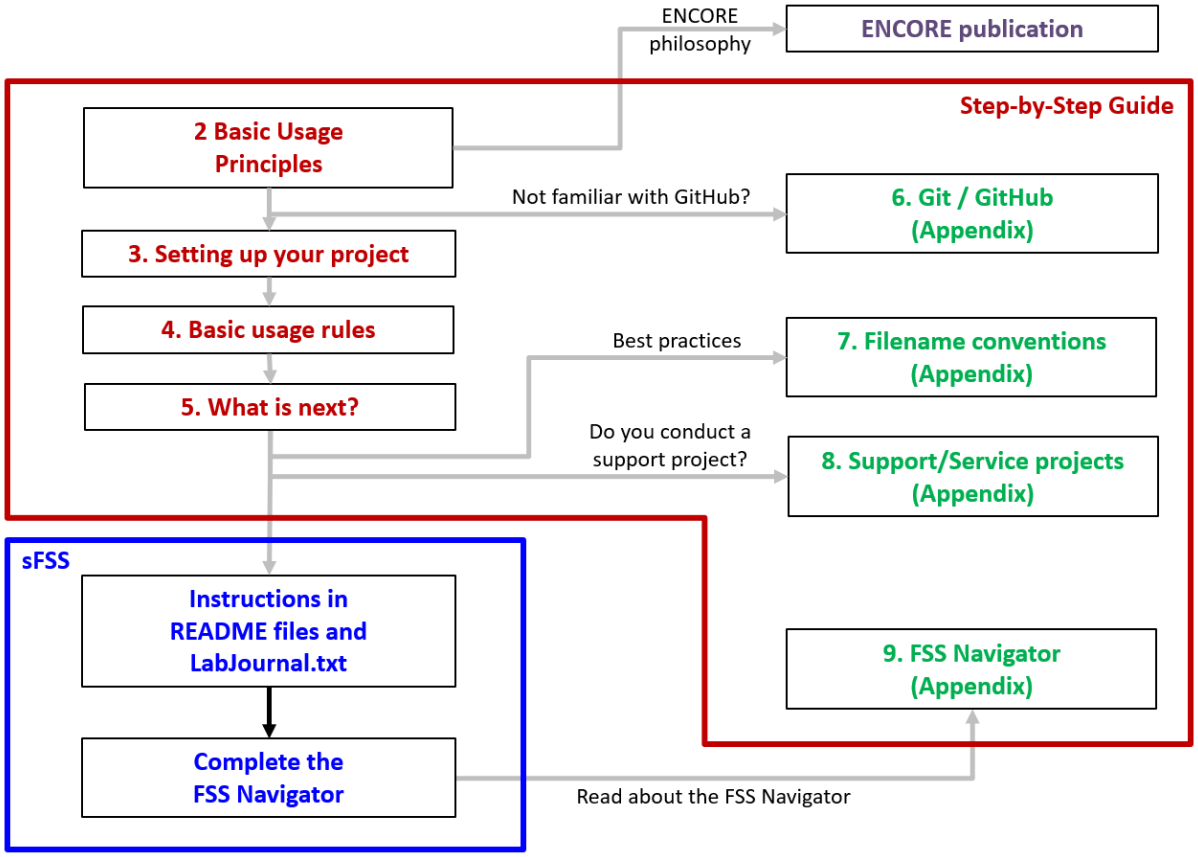
3. **The Compendium Navigator**: this provides a simple web-based navigation page, which provides a guide for peers that aim at inspecting the overall structure and content of a project (*Section 9; Appendix).*



Supplementary Figure 1. The main parts of the standardized File System Structure (sFSS) template. Project corresponds to the root directory of the template. Other blocks are sub-directories. Data can be organized at several levels depending of the preference for the project. Processing contains one or more computations (e.g., pre-processing, statistical analysis, simulation), which includes code and results. Only code and code documentation are synchronized with the GitHub project repository. All parts of the template contain pre-defined files (e.g., README, Lab Journal) to document each part of the project.

1.2 HOW TO GET STARTED?

Supplementary Figure 2 shows the section to read (red) from this Step-by-Step guide. When needed you can consult the Appendices for additional information. Once you have set up your project you will find additional instructions in the various README Markdown files, the LabJournal.txt, and some other files. Although it seems a lot of work, once you have gone through the procedure once, it will take you about 15-30 minutes to set up a new project.



Supplementary Figure 2. This flow diagram guides you through the documentation. The sections inside the red box are part of this Step-By-Step guide. The sections indicated in red should be read and followed. The sections indicated in green are optional. Once you have set up your project, you will find more specific instructions inside the sFSS (blue box).

2 BASIC USAGE PRINCIPLES: TRANSPARENCY AND REPRODUCIBILITY

This section describes the most important general principles to follow to ensure transparency and reproducibility of your project. A more comprehensive discussion is found in the ENCORE paper (*see References*).

1. **Principle 1. The sFSS project should be self-contained.**
    - **Avoid external documentation**. All relevant project documentation should be kept inside the sFSS. Consequently, don't keep relevant documentation or information/discussions in email archives, paper notes, Slack, WhatsApp, etc. Any relevant information should be described within the sFSS (e.g., Lab Journal or README files).
    - **Use relative references/links** to any item (data, code, results) in the sFSS. Relative links (with respect to the sFSS root) also ensures that code remains executable if the sFSS is copied to a different location (by your peers).

2. **Principle 2. Put the documentation where it belongs.** Keep the different parts of the documentation in the data, software, and results subdirectories, instead of having one large file (e.g., the lab journal, PowerPoint presentation, or pre-manuscript) that contains all documentation.
    - This makes it easier to find and maintain.
    - Make cross-references between the different parts of the documentation when necessary. For example, link descriptions of results to used code and data.
    - You can copy (parts of) this information to (PowerPoint) presentations or your manuscript when needed. In general, the sFSS will contain more detailed information than found in (the supplement) of a publication.

3. **Principle 3. Explain to your peers.** ENCORE aims to improve transparency and reproducibility. Specifying the conceptual information in detail will also help to identify methodological problems. Use the sFSS to teach your peers what you did and why you did it.
    - **Conceptual information** should provide information the concepts and approaches used in the project to increase transparency and to enable full understanding of the computational project by your peers. It can be related to the data, code, and results.
        - For example, describe the applied computational methods, an explanation of why and how the computational experiments were done, how data were acquired in the wet-lab, why and how the data were pre-processed, observations that you make about the data and results, relevant thought processes, etc. References to relevant resources should be supplied as much as possible.
    - The sFFS and its (pre-defined) files (e.g., LabJournal.docx, 0_PROJECT.md, and 0_README.md) should enable your peers (both within and outside your research group) to understand and reproduce your project. Document your project with this in mind.
    - To specify the conceptual information, use the README Markdown files or use other file types (e.g., Word, LaTeX) if that is more convenient. Link different documentation files whenever needed.

4. **Principle 4. Keep the documentation up to date**. Update your sFSS and its pre-defined files on a continuous basis during the project. Don't attempt to do this at the end of the project.
    - Keeping the project up to date allows your supervisor and/or collaborators to always follow and contribute to the project.
    - It is impossible to recall all project details from memory and document these once the project is finished, published, or archived.

3 SETTING UP YOUR PROJECT: THE RECIPE

If you are new to GitHub, then first read (*Section 6 ; Appendix*) and make sure you have a **GitHub account** and to **install Git Bash** if you are working on Windows. The recipe below will take you step by step towards the creation of the sFSS and a GitHub repository for a new project, and the setup of the Compendium Navigator. There are alternative ways of doing this, but this will get you going.

ENCORE AUTOMATION. Note that the GitHub repository https://github.com/EDS-Bioinformatics-Laboratory/ENCORE_AUTOMATION provides a Windows batch file and Unix/MacOS bash script (CREATE-ENCORE-PROJECT.{bat,sh}) to automatically setup an ENCORE project: (i) Create a local project directory, (ii) create a GitHub repository, (iii) connect the project directory to the GitHub repository (steps 3.1.1, 3.2.1, 3.2.2, and 3.2.3 below). Note that step 3.1.2 has to be performed manually and that you may want to adjust the .gitignore file to your own preference

3.1 STEP 1: CREATE AND INITIALISE THE STANDARDIZED FILE SYSTEM STRUCTURE (SFSS)

The next step is to create a project directory on your computer that is based on the sFSS template located in the ENCORE GitHub repository.

Note about the Markdown files.

In the sFSS you will find Markdown files (file extension .md). If you are not familiar with markdown then visit https://www.markdownguide.org/getting-started. Markdown files can be edited with any text editor but are better visualized in a Markdown viewer such as Typora (www.typora.io; Windows, Mac) or Notepad++ (Windows; install the MarkdownViewer++ plugin).

3.1.1 CREATE THE SFSS TEMPLATE

1. Start git bash in the directory where you want to create your project.

2. Create a project directory (e.g., **20231201_Project)** containing the sFSS template from the ENCORE GitHub repository using the next command in git bash:
   - git clone https://github.com/EDS-Bioinformatics-Laboratory/ENCORE_**20231201_Project**

3. Move into the directory **20231201_Project** and remove the file README.md and remove the sub-directory .git (which may be hidden on your system).

3.1.2 INITIALIZE THE SFSS PROJECT

4. Enter the required information in the **0_PROJECT.md** file in the sFSS root directory
   - See instructions in this file.
   - This file is used by the Compendium Navigator (see below).

5. Provide information about the software and hardware environment.

- See 20231201_Project**\Processing\0_SoftwareEnvironment\0_README.md** for instructions.
- Provide the information already available to you, update whenever needed.

6. Start your lab journal
   - See 20231201_Project**\ProjectDocumentation\LabJournal.txt** for instructions**.**

3.2 STEP 2: SETUP YOUR GITHUB REPOSITORY AND CONNECT TO THE SFSS

3.2.1 CREATE A GITHUB REPOSITORY

1. Go to your GitHub account at https://github.com.

2. Click on the 'New' button.

3. Choose a repository name.
   - Use an informative name (e.g., **B-cell_DiversityAnalysis**).
   - Don't put the date in the name.
   - Don't put a literature reference in the name.

4. Add a description. Note, the description should start with the name(s) of the repository owner.
   - Example: "Antoine van Kampen, Barbera van Schaik: Analysis of B-cell repertoires".
   - *Rationale:* The number of repositories may grow quickly if you do many projects. By starting with the owner's name in the 'About' field you can directly see to who a project belongs.

5. Make the repository 'Private'.
   - *Note*: at a later stage you can still decide to make a repository Public.

6. Do not add the default GitHub README file.

7. Do not add .gitignore

8. Choose the GNU General Public License v3.0
   - *Rationale*: this open-source license allows other people to use, extend, and modify our software. These changes will again become open source (of course if you decide to share your software).

9. Now 'Create repository'
   - You should now see your new repository with only a single file (LICENSE) and a single branch (main).

10. Click on the 'wheel' ('cog') right of About and add Topics.
    - Standard keywords are: research, support, education.
    - Look at other repositories for used keywords to be consistent.
    - *Rationale:* having keywords helps to retrieve specific repositories.

3.2.2 CONNECT THE SFSS PROJECT TO THE GITHUB REPOSITORY

11. Enter information in the 20231201_Project**\Processing\README.md** in the sFSS.
    - See instructions in this file.
    - This **README.md** file is the default README file that is used by GitHub (and will be synchronized with the repository in the next steps).

12. Edit github.txt
    - Add link to your GitHub repository (e.g., https://github.com/YourAccount/B-cell_DiversityAnalysis.git**)** in 20231201_PROJECT**\Processing\github.txt** in the sFSS.
    - This file is used by the Compendium Navigator (see below).

13. Create a .gitignore file
    - In the 20231201_PROJECT**\Processing\.gitignore** you can specify the files and directories that should not be synchronized with your GitHub repository.
    - The gitignore should be configured such that only code and documentation are synchronized with GitHub. Not the, for example, data or results.
    - See instructions in 20231201_Project**\Processing\README.md.**

3.2.3 SYNCHRONIZE YOUR SFSS PROJECT WITH YOUR GITHUB REPOSITORY

Now you have done some basic administrative work, you are ready to synchronize part of your project (20231201_Project) with your GitHub repository (https://github.com/YourAccount/B-cell_DiversityAnalysis.git).

Following the ENCORE philosophy that the sFSS is self-contained (and is the entity shared with peers), we only synchronize code and code documentation with GitHub to allow tracking of software and documentation versions.

If you configured the 20231201_PROJECT**\Processing\.gitignore** correctly, then only code and documentation will be synchronized with GitHub.

1. Go to 20231201_PROJECT**\Processing**
2. Start Git Bash in this directory (Microsoft Windows: right mouse click, then select 'Open Git Bash here'. MacOS: git is installed by default and can be used after opening the Terminal)
3. Enter the following git commands (after each command you can use git status to check):
    - git init --initial-branch=main # note the use of the double dash
    - git remote add origin [*URL of repo*]
        - URL of repo: as entered in github.txt, e.g.,
        - https://github.com/YourAccount/B-cell_DiversityAnalysis.git
    - git pull origin main
    - git add .
    - git commit -m "First sync" -m "First sync with GitHub after setting up the sFSS"
    - git push -u origin main

if you go to https://github.com/YourAccount/B-cell_DiversityAnalysis.git in your web browser then you see that part of the sFSS is synchronized with your repository.

Note that the command 'git init' has created the (hidden) directory .git in your Processing directory. Don't remove it.

3.2.4 KEEP YOUR GITHUB REPOSITORY UP TO DATE

From now on you can use the following git commands to keep your project directory and GitHub synchronized (preferably, you do this on a continuous basis).

1. Go to 20231201_PROJECT**\Processing**.
2. git pull https://github.com/YourAccount/B-cell_DiversityAnalysis.git
   - Only perform this command if there were changes (from your collaborators) on the GitHub repo that are not yet in your local repository (in .git)
3. git add .
4. git commit -m "short description" -m "long description"
5. git push

3.3 STEP 3: SETUP THE COMPENDIUM NAVIGATOR

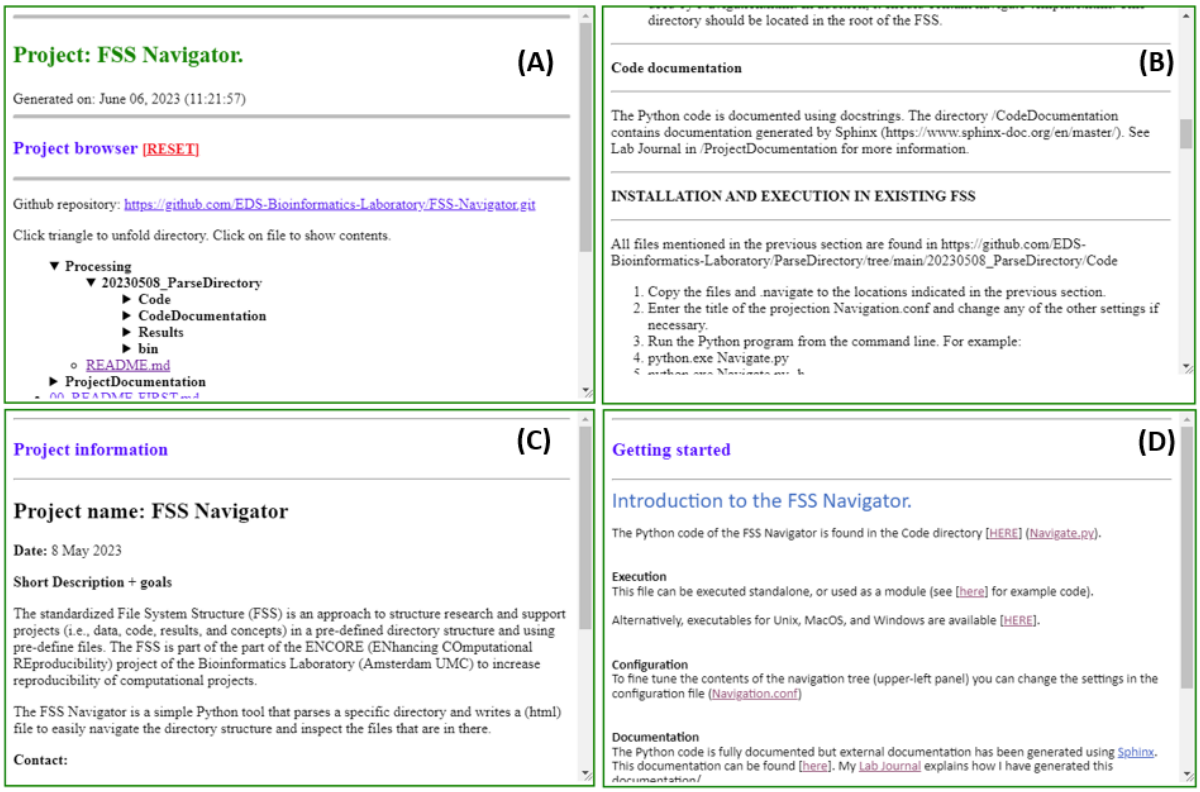
The Compendium Navigator is a small Python program that creates a HTML file (20231201_Project**Navigate.html**), which you can open in your web-browser to inspect the sFSS. *Supplementary Figure 3* shows an example of an **Navigate.html** reflecting the Compendium Navigator project itself. To setup the Compendium Navigator, proceed as follows:

1. The Compendium Navigator uses 20231201_Project**0_GETTINGSTARTED.html**. Open this file in your browser to get further instructions. Typically, you will do this near the end of your project.

2. Configure the Compendium Navigator by changing the parameters in the configuration file (20231201_Project**\Navigation.conf**). This file should also contain the title of the project.

3. Finally, generate the 20231201_Project**Navigate.html** file by executing the Python program (20231201_Project**\Navigate.py**) or one of the available executables. See 20231201_Project**\README.md** for instructions.

See *Section 9 (Appendix)* for further details.



Supplementary Figure 3. **Compendium (FSS) Navigator for the Compendium Navigator project**. Web-browser showing Navigate.html for the Compendium Navigator project. (A) Expandable sFSS directory tree and link to the project's GitHub repository. The project team can configure which directories and files to show. (B) Content of selected file. In this example, the panel show the content of the default GitHub README markdown file. (C) General project description, contact person, and collaborators (0_PROJECT.md). (D). Getting started explains the project and includes links to the various files and directories in the sFSS (0_GETTINGSTARTED.html).

4 BASIC USAGE RULES OF THE SFSS TEMPLATE AND THE PRE-DEFINED FILES

4.1 ENCORE SFSS TEMPLATE

- **Template source.** For new projects the latest ENCORE sFSS template should be downloaded from GitHub (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE).

- **GitHub Releases.** Based on use cases and experience, ENCORE is expected to further improve over time resulting in minor updates in the sFSS structure, the pre-defined files, the Compendium Navigator, and/or this Guide document. Changes in the sFSS structure will only be implemented when necessary. In principle, always use the latest template from the specified source GitHub. Current and previous (pre-)releases will always be available as a GitHub Release. GitHub Releases will include the Compendium Navigator binaries. Changes between versions are documented in GitHub Issues and with the specific releases.

- sFFS Navigator updates. Updates (e.g., bug fixes) of the Python code and executables are available from the Navigator repository Releases (https://github.com/EDS-Bioinformatics-Laboratory/FSS-Navigator/releases). Compatibility, with specific ENCORE templates is documented with each Release. Navigator versions and Compatibility with ENCORE template versions is also available from the ENCORE wiki (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/wiki/sFSS-Navigator).

- **Automatic instantiation.** In case you develop/use software to automatically instantiate a new ENCORE project, then this software should retrieve the latest ENCORE template, or itself generate the sFSS structure including the pre-defined files (and their content). In any case, make sure that the instantiated template is identical to the default template from the specified template location.

- **Directories and files.** Unless the instructions in this Guide or the README files tell you otherwise, the sFSS directory names and structure, and the pre-defined file names should not be changed to ensure consistency across projects. In specific
    - Don't add/remove dashes, spaces.
    - You are allowed to remove the '0_' prefix from the README files if you do not want it to be at the top of the file list. However, don't change the name of the default GitHub README.md in \Processing.
    - Additional subdirectories may be added to the sFSS if needed to get a better organization of the project, but the overall base structure and use should remain as is. For example, you can add a sub-directory \Figures in \Results
    - Files (e.g., Step-by-Step ENCORE guide) that are not relevant for the Compendium Recipient (e.g., peer, reviewer, archive) should be removed. Additional, directories and files that are not used can be removed but don't remove the following files in the sFSS root directory:
        - README.md
        - 2_CITATION.md
        - 3_LICENSE.md

- **README markdown files**. Most directories contain a 0_README.md Markdown file. These files are used to clarify the content of the various directory in detail.

- Each README file contains **instructions** in *italics*. These instructions can be removed once you completed the README file.
- Most README files contain a basic **template** to guide you in providing the required information. However, if necessary, provide additional information to **ensure reproducibility and transparency**.
- If you prefer you can use any other **(additional) format** like Microsoft Word or LaTeX. If you do, then also provide PDF files for each of these documents.
- Do not change the file name or format of the README.md in the \Processing directory since this is the default README file of GitHub.

4.2 GENERAL

- **Project name.** Project name may contain a prefix such as year, month, day (YYYYMMDD_ProjectName) or project identifier (ID_ProjectName).
- **File names.** Name all project files in such a way that the name reflects their content or function.
- **Person names.** If you use names of persons in any part of your documentation then ensure that their first and last name, titles, affiliation, and project roles are documented. Otherwise, your peers might not know who these persons are. If the person is a collaborator on the project, then (s)he should be added to the 0_PROJECT.md file found in the top-level directory.
- **sFSS is point of entry.** The sFSS is the entry point of a project, not GitHub. Consequently, all project information (e.g., background, data, code, results) should be stored within the sFSS. GitHub is only used for software versioning. The sFSS should only contain the (latest) software version(s) that were used to produce the results present in the sFSS. This, however, does not exclude the possibility to also share the GitHub repository with your peers.
- **Backup.** Make (incremental) backups of your project daily (e.g., to an external hard disk and/or Cloud).

4.3 EXTERNAL (BIG) DATA AND EXTERNAL COMPUTING INFRASTRUCTURES

There might be situations where the data is stored at a location outside the sFSS (i.e., not in one of the \Data directories). For example, in case of very large data files or in case of existing data repositories, it might be impossible or undesirable to permanently store the data within the sFSS. In these situations, it is considered best practice to setup the required sFSS data (sub)directories as you would normally do but only temporarily copy the data inside the sFSS at the moments the data is needed for the computations. In this way, all paths to the data can remain relative to the root of the sFSS instead of having paths pointing to external resources that might not be accessible for your peers. For sharing the sFSS project with your peers you may decide to make a copy containing the full data or, alternatively, document within the \Data directory how the data can be obtained. In any case, data copying should not incorporate any manual step to change the data.

There might also be situations in which, for example, the sFSS is stored on your laptop but computations are done on another (high performance) computing infrastructure. In such cases it is considered best practice to copy the complete sFSS to the other computing infrastructure to perform the computations and collect the results and, subsequently, copy back the sFSS to your laptop (now including the results from your computations.

Finally, you may have scenarios in which you don't have a permanent copy of your data inside the sFSS and use external computing infrastructure. In these cases, you can combine the principles outline in the previous two paragraphs.

4.4 SHARING YOUR SFSS

The sFSS is meant to be shared with peers who want to reproduce are built upon your project. Since the sFSS is self-contained (contains all data, code, results, and documentation), it can be zipped and send to your peer, or stored in a public repository such as Zenodo.

However, ensure that you are allowed or want to share all the information within the sFSS. You might not want to share, for example, non-open access publications (pdf files), patient data, or new research ideas.

Since all relevant code is stored within the sFSS, there is no direct need to share the corresponding GitHub repository by making it public. This is up to you.

Support (service) projects are a type of projects conducted for third parties. For example, the analysis of a single-cell transcriptomics dataset produced by an experimental wet-lab biologist to study the role of B-cells in microbe infections. The sFSS of support projects can be shared with the customer in a way explained above. However, since the customer will likely only be interested in the results of the computation you may decide to only share the results. You may also decide to restructure the sFSS prior to sharing to make is more manageable for the customer. If you restructure and/or share parts of the sFSS you can place this in the \sharing directory such that you can always keep track of the information that was shared.

5 WHAT IS NEXT?

Congratulations! You have now set up the three main components of ENCORE: a dedicated sFSS, a corresponding project GitHub repository, and the Compendium Navigator.

There are a few steps left to take:

1. Read the ENCORE publication (*see Section 1*) to get a better understanding of the ENCORE philosophy.

2. Read the README.md file in the root of the sFSS

3. Browse through the various sFSS directories and consult the 0_README.md files, the LabJournal.txt, and other files for specific instructions about the information you need to provide in each sub-directory.

4. Populate the sFSS with project information, data, and code you may already have available.

5. **KEEP THE sFSS UPDATED CONTINOUSLY!!**

Complementary Tools
Note that ENCORE does not rely on specific tools except for Git/GitHub for code versioning. However, to improve reproducibility, it is important to use complementary tools. For this we refer to:
- **Software Environment**: see 0_README.md in \Processing\0_SoftwareEnvironment
- **ENCORE WIKI**: https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/wiki/Complementary-Tools
- **Software engineering**: see 0_README.md in \Processing\NameOfComputation_1\Code

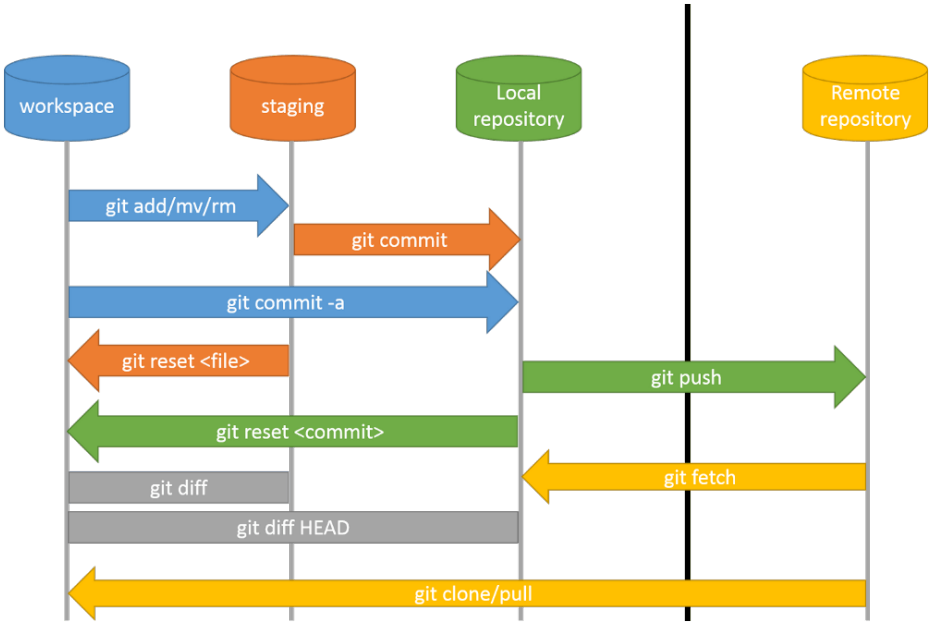
6 APPENDIX. USING GITHUB AND GIT

Disclaimer. It is not the intention of this Guide to give a full overview of all git/GitHub scenarios and commands. However, you may find the information below useful in case you run into problems in using GitHub and Git in the context of the sFSS.

Git is a free distributed **version control** system suitable for tracking modifications in source code during software development. It was originally created as an open-source system for coordinating tasks among programmers, but today it is widely used to track changes in any set of files.

GitHub is a web-based **Git repository**. This hosting service has cloud-based storage. GitHub offers all distributed version control and source code management functionality of Git while adding its own features. It makes it easier to collaborate using Git. GitHub repositories are open to the public. Developers worldwide can interact and contribute to one another's code, modify, or improve it.

Think of Git as a single computer and GitHub as a network of multiple interconnected computers, all with the same end goal but a wildly different role for how to get there (*Supplementary Figure 4*)



Supplementary Figure 4. The overall architecture of the git/GitHub environment.

6.1 GITHUB ACCOUNT

If you do not yet have a GitHub account then visit their website at https://github.com, and sign up for GitHub.

6.2 INSTALL GIT BASH

Note: MacOS has git installed by default and can be used after opening a Terminal.

1. Download Git Bash: https://git-scm.com/downloads
2. Optionally you can download one of the GUI clients but using git bash (command line) will get you a better understanding of git.
    - GUI client: https://desktop.github.com/

Git Bash allows you to access GitHub from your own computer/laptop.

6.3 GIT DOCUMENTATION

There is a lot of documentation and tutorials on the internet.

- GitHub Docs (https://guides.github.com/)
- Short GitHub introductory videos
    - https://www.youtube.com/watch?v=nhNq2kIvi9s
    - https://www.youtube.com/watch?v=USjZcfj8yxE
- GitHub cheat sheet: https://education.github.com/git-cheat-sheet-education.pdf
- More about .gitignore
    - https://git-scm.com/docs/gitignore
    - https://github.com/github/gitignore (templates)

References

- Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A Quick Introduction to Version Control with Git and GitHub. PLoS Comput Biol, 12(1), e1004668.
- Perez-Riverol, Y., Gatto, L., Wang, R., Sachsenberg, T., Uszkoreit, J., Leprevost Fda, V., . . . Vizcaino, J. A. (2016). Ten Simple Rules for Taking Advantage of Git and GitHub. PLoS Comput Biol, 12(7), e1004947.
- Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. Source Code Biol Med, 8(1), 7.

6.4 GITHUB AND GIT: STARTING FROM SCRATCH

If you are completely new to GitHub and Git then you can follow the next steps to create your first repository.

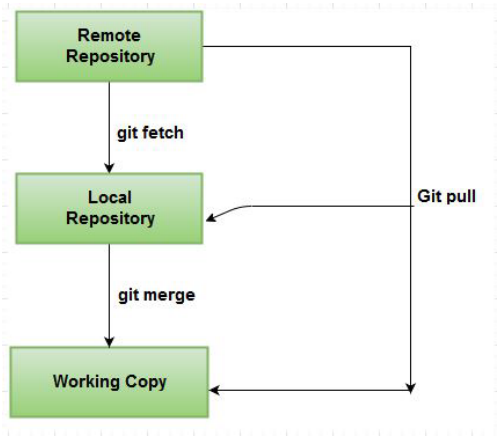
1. Go to https://github.com/ to create your own account
2. At a certain stage you will need a Fine-Grained personal access token to access your repositories. Read all about it: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token
3. Create a new public or private repository on GitHub. Give it a name and short description but do not add any files (e.g., README.md, LICENSE, .gitignore) to it.

4. Once you created the repository you will see the Quick Setup screen that also shows the name of the repository that you created. This looks something like: **https://github.com/YourAccount/test.git**
5. Next create a directory on your computer that should mirror your repository locally, and step into it, and add a Markdown file README.md.
6. Start Git bash in this directory
7. Create your local repository:
   - git init --initial-branch main #note the double dash #note the double dash
8. Add Git credentials for this specific repository (or use the -global option to do this for all current/future repositories)
   - git config credential.helper manager-core
   - git config user.name YourUserName
   - git config user.email YourEmail
   - The next time you will commit/push to the repository for which you added the credentials, Git will ask you for the credentials for that particular remote server if it is unable to find the username and password already stored.
9. Next give the following commands:
   - git remote add origin **https://github.com/YourAccount/test.git**
   - git remote -v #check, or use git remote set-ulr [ulr.git] to change
   - git add .
   - git commit -m 'First sync' -m 'This is the first syncrhonization of my local repositories'
   - git push --set-upstream origin main
10. Note: if you get a 'fatal error' in step 9 then it is likely that something went wrong with the authentication.
11. Go back to GitHub in your web browser and select your Repository. You will see that the README.md file is added, and its contents is shown by default on the main page.
12. Click on the 'Cog' icon to change your description, add topics (keywords), and add your (personal) website.
13. Next, select 'Add file' and 'Create new file'. Type 'LICENSE' as the file name. This will activate a button on the right part of the screen where you can select a License template. Select one, Review and Submit, and (don't forget) to Commit at the bottom of the screen (select 'commit directly to the main branch').
14. Now we need to synchronize these changes with your local repository:
    - git pull
15. This is basically it. The next time you add files to your local directory you only have to give the following commands to update your remote github repository:
    - git add .
    - git commit -m 'I made a change'
    - git push

6.5 FURTHER GIT/GITHUB NOTES

6.5.1 GIT PULL VS GIT FETCH

git fetch is similar to git pull but doesn't merge, i.e., it fetches remote updates (refs and objects) but your local stays the same (the origin/master gets updated but master stays the same). git pull pulls down from a remote and instantly merges (*Supplementary Figure 5*).



Supplementary Figure 5. Git fetch vs Git pull.

To check for differences between your remote repository and the local working copy:

git fetch

git diff main origin/main

If you are happy with the changes then you can merge with git merge or just do a git pull

See also: https://stackoverflow.com/questions/292357/what-is-the-difference-between-git-pull-and-git-fetch

6.5.2 USE OF BRANCHES

Note: the use of the 'Master' branch is discouraged. Instead, you should use 'Main'. For some more background about this see

- https://medium.datadriveninvestor.com/why-githubs-change-from-master-to-main-is-not-the-solution-a3ac38cc48dd
- https://stevenmortimer.com/5-steps-to-change-github-default-branch-from-master-to-main/

The default branch in git is the **"main" branch**. Common practice is that this is the stable code. When you would like to develop a new feature, fix a bug, etc. you can make use of branches. This code will live next to your stable version and once you are satisfied with the changes you have made (and when you have stable code again) you can merge the newly developed feature into your master branch.

6.5.2.1 OVERVIEW OF THE BRANCHES ON YOUR WORKSTATION

git branch - will show you the list of branches. There is an asterisk (*) in front of the active branch on your workstation.

6.5.2.2 CREATE A NEW BRANCH

When you have several branches, you probably would like to create a new branch with the 'main' branch as a starting point most of the time. First switch to the 'main' branch and then create a new branch.

git checkout main - switch to the main branch
git branch devel-some-feature - a new branch will be created with the name 'devel-some-feature'

Note that you are not in this branch yet when you create a new one. Switch to this branch with:
git checkout devel-some-feature - go to the new branch

You can start developing. Even when you are not ready yet you can commit and push your code to the repository on GitHub because this branch is separate from your stable main branch.
git commit -m 'some description' - make a snapshot of your changes, it will be logged with a git version
git push origin devel-some-feature - send your snapshot to GitHub

6.5.2.3 MERGE THE NEW CODE INTO THE MASTER BRANCH

In the case that you are satisfied with the changes/new feature and you have tested whether the code works as it should you can merge it in the master branch. The order is as follows: first you switch to the main branch, then you merge the new developments into the master branch.
git checkout main - switch to the main branch
git merge devel-some-feature - merge the new feature into your main branch

Git will check whether there is conflicting code and notify you when this is the case. When that happens the merging process will stop and you will get the opportunity to resolve this. As soon as you are happy and everything works you can commit and push all the changes in the main branch.

git commit -m 'describe the changes' - make a snapshot of your changes on your workstation
git push origin main - synchronize with GitHub

In the case that there are no conflicts git will just merge the changes. It is still good to test your code again and then push it to GitHub.

Resolving conflicts: www.atlassian.com/git/tutorials/using-branches

Tips for collaboration and best practices: www.atlassian.com/git/tutorials

6.5.2.4 MOVING MASTER TO MAIN

https://www.r-bloggers.com/2020/07/5-steps-to-change-github-default-branch-from-master-to-main/

git branch -m master main

git push -u origin main

Sometime there might be a 'mixup' of branch names (main vs master). If so, these can be resolved with the following commands:

- → git rm –cached -r .
- → git branch main
- → git checkout main
- → git merge master
- → git push origin main
- → if necessary: git push origin --delete master

6.5.3 USING .GITIGNORE

In the file .gitignore (which lives is /Processing) you can configure which files and/or directories will not be synchronized with the GitHub repository.

What files should be ignored?

Ignored files are usually platform-specific files or automatically created files from the build systems. Some common examples include:

- Runtime files such as log, lock, cache, or temporary files.
- Files with sensitive information, such as passwords or API keys.
- Compiled code, such as .class or .o.
- Dependency directories, such as /vendor or /node_modules .
- System files like .DS_Store or Thumbs.db
- IDE or text editor configuration files.

In addition, for the standardized sFSS only code, notebooks, and code documentation should be synchronized with the GitHub repository. Thus, the .gitignore file should exclude at least

- Analysis output (e.g., tables and figures)
- Data

You can use git status --ignored to check which files and/or directories are ignored.

Alternatively, you can use git check-ignore -v [path/file] to show directories and/or files that are excluded from the repository. The -v option also returns the exclude pattern from the .gitignore file. However, this does not always give the required output (see https://stackoverflow.com/questions/40763820/git-check-ignore-output-empty-but-still-being-ignored).

6.5.4 AUTHORIZATION

For GitHub authorization issues see:

- https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/
- https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token

6.5.5 PROBLEMS WITH 'MERGING'

In case you have problems with mering files (after conflicting copies) you can use:

git mergetools

6.5.6 REMOVE ALL FILES IN A GITHUB REPOSITORY

```
cd /tmp                          #make temporary directory
git clone /your/local/rep        # make a temp copy
cd rep
git rm -r *                      # delete everything
git status                       # everything but those copied will be removed
git commit -a -m  'deleting stuff'
git push
```

6.5.7 HOW TO USE A GITHUB REPO WITH RSTUDIO?

See, for example, https://happygitwithr.com/rstudio-git-github.html. Many other useful tips regarding R, RStudio and GitHub can be found here as well.

7 APPENDIX. FILENAME CONVENTIONS

7.1 GENERAL CONVENTIONS

- A good format for date designations is YYYYMMDD. This format makes sure all of your files stay in chronological order, even over the span of many years.
- Try not to make file names too long, since long file names do not work well with all types of software.
- Special characters such as ~ ! @ # $ % ^ & * ( ) ` ; < > ? , [ ] { } ' " and | should be avoided.
- When using a sequential numbering system, using leading zeros for clarity and to make sure files sort in sequential order. For example, use "001, 002, ...010, 011 ... 100, 101, etc." instead of "1, 2, ...10, 11 ... 100, 101, etc."
- Do not use spaces. Some software will not recognize file names with spaces, and file names with spaces must be enclosed in quotes when using the command line. Other options include:
    - Underscores, e.g., file_name.xxx
    - Dashes, e.g., file-name.xxx
    - No separation, e.g., filename.xxx
    - Camel case, where the first letter of each section of text is capitalized, e.g., FileName.xxx
- Avoid using spaces and other symbols in your filenames. Dashes and underscores are allowed.

7.2 NAMING VERSIONS

When creating new versions of your files, record what changes are being made to the files and give the new files a unique name. Consider the following:

- Include a version number, e.g "v1," "v2," or "v2.1".
- Include information about the status of the file, e.g. "draft" or "final," as long as you don't end up with confusing names like "final2" or "final_revised".
- Include information about what changes were made, e.g. "cropped" or "normalized".

7.3 SOFTWARE VERSIONING

To keep track of different versions (and releases) of the software you may consider to use a software versioning scheme. For more information see:

- Versioning: https://en.wikipedia.org/wiki/Software_versioning
- Semantic versioning: https://semver.org/
- How to manage version numbers in git: https://stackoverflow.com/questions/37814286/how-to-manage-the-version-number-in-git

This would allow to connect specific software versions to results in your sFSS.

8 APPENDIX. SUPPORT PROJECTS

Support (service) projects are a type of projects conducted for third parties. For example, the analysis of a single-cell transcriptomics dataset produced by an experimental wet-lab biologist to study the role of B-cells in microbe infections.

For such support project the ENCORE principles are directly applicable. However, one should keep in mind that the documentation of the project inside the sFSS is from the perspective of the data analysis, and not from the perspective of the biological question.

Thus, one would typically document the computational approaches used to analyse the data. The \Literature directory might contain the publications explaining these methods in detail. In addition, one would use the documentation obtained by the biologist to describe the samples, experimental design, experimental protocols, etc. But also, the objective of the data analysis, or the hypothesis to be rejected.

Support projects may comprise routine analysis that are frequently performed for different customers. In this scenario, one may setup dedicated sFSS templates that contain the generic part of the documentation (e.g., description of the data analysis pipeline to analyse single-cell transcriptomics data).

8.1 USING GITHUB BRANCHES FOR SUPPORT

Following the ENCORE guidelines, one would make a new GitHub repository for each new project. However, for frequently performed analyses one would get many repositories with almost the same code (since only code and documentation is synchronized with GitHub). Therefore, one could decide to store the code only once in GitHub. However, then changes (e.g., different parameters, filenames, experimental design) made for specific projects can not be tracked.

A better alternative is to use GitHub branches for each new project requiring the same type of analysis. The main branch will contain the reference code. Next, for each new project you create a new branch with a descriptive name (e.g., SingleCellAnalysis_May2023_JohnDoe). The (modified) code for this project can then be pushed to this branch. This branch will not be merged with the main branch. In the sFSS you document which branch was used for the project, and what changes were made to the code.

9 APPENDIX. THE COMPENDIUM NAVIGATOR

The sFSS may eventually, at the end of a project, contain a large amount of information potentially making it difficult for peers to determine the best point of entry. The Compendium Navigator was developed to provide a first guidance through the project. The Compendium Navigator itself was developed following the ENCORE approach, and the project package is found in Zenodo (https://doi.org/10.5281/zenodo.7985655). The Compendium Navigator is a Python program (20231201_Project**Navigate.py**) that creates a html file (20231201_Project**Navigate.html**), which you can open in your web-browser to inspect the sFSS. In addition to the Python program, executables for Windows, Mac OS and Unix are provided to ensure the navigator can be used if Python is not installed. The generated web page consists of four panels (*Supplementary Figure 3*). The content of the panels is configured by the project owner, which allows to guide peers to the important parts of the project. The navigator also provides a convenient tool during the project to, for example, browse and show results. Note, that the Compendium Navigator is work in progress and updates will become available in the future.

Note that the executables are not part of the active GitHub repository, but are part of the GitHub releases.

Notes

UTF-8 encoding

The Compendium Navigator (Navigate.py) converts the README.md Markdown files to HTML. It assumes a Unicode encoding (i.e., UTF-8) of the characters in the README file. UTF-8 (Unicode Transformation Format – 8-bit) is a variable-length character encoding standard used for electronic communication and extends ASCII.

If the README file uses an incorrect encoding then you will see an error like this when executing Navigate.py:

> *File "C:\Users\ahcva\anaconda3\envs\BIO\lib\encodings\cp1252.py", line 23, in decode*
> *return codecs.charmap_decode(input,self.errors,decoding_table)[0]*
> *UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position 2308: character maps to <undefined>*

This can happen, for example, if you copy from a Word document to the README Markdown file. However, visual inspection of the readme file may not directly reveal the incorrect character(s).

To fix the Markdown file:

1. Open file in Notepad++
2. Check the encoding in the 'Encoding' menu
    a. If the encoding is not set to UTF-8 then 'Convert to UTF-8'
    b. If the encoding is set to UTF-8 then set encoding to 'ANSI' and, subsequently, 'Convert to UTF-8'
3. Inspect the README file for 'strange' characters and fix.

Python environment

In order for Navigate.py to execute it requires Python to be installed. In addition, it requires that used Python packages are installed. Instructions are found here: https://github.com/EDS-Bioinformatics-Laboratory/FSS-Navigator. The 'requirement' files to setup a Python environment are found here: https://github.com/EDS-Bioinformatics-Laboratory/FSS-Navigator/tree/main/0_SoftwareEnvironment

10 APPENDIX. DOCUMENT VERSION HISTORY

8 March 2023

- Added the section 'VERSION HISTORY'
- Added the section "GitHub and Git from scratch".

29 March 2023

- Added section 'Git fetch vs Git pull
- Changed of title page (using ENCORE for the time being)
- Started further changes/improvements to the document to accompany the publication about this initiative

14 April 2023

- Added the section 'Using .gitignore'

28 July 2023

- Re-written and re-structured this Guide, and improved layout.
- Added an introduction.
- Added more general information about ENCORE/sFSS.

19 February 2024

- Minor changes.

12 June 2024

- Several small changes
- Now refers to repository Discussion and Wiki, and to ENCORE AUTOMATION
- In the instructions for setting up an ENCORE project and GitHub repository, it now explains which steps are taken care of by the scripts in ENCORE AUTOMATION.
- We explained the compatibility between Compendium Navigator and ENCORE template versions and refer to the WIKI.
- We clarified that files that are not relevant for the Compendium Recipient can/should be removed.
- Addition of a remark about the use of complementary tools

30 July 2024

- Updated instructions to run Navigate.py
- Replaced (s)FSS Navigator with Compendium Navigator
- Updated Appendix 9 (Compendium Navigator)
- Various minor changes.

11 APPENDIX. ACKNOWLEDGMENTS

ENCORE is an initiative of the Bioinformatics Laboratory (www.bioinformaticslaboratory.eu) with contributions from many group members:

- Prof. dr. Antoine van Kampen
- Dr. Perry Moerland
- Dr. Aldo Jongejan
- Dr. Adrie Dane
- Barbera van Schaik
- Eric Wever
- Dasha Balashova
- Rodrigo Garcia Valiente
- Danial Lashgari
- Utkarsh Mahamune
- Mia Pras-Raves

12 SUPPLEMENTARY REFERENCES

1. Antoine H.C. van Kampen, Utkarsh Mahamune, Aldo Jongejan, Barbera D.C. van Schaik, Daria Balashova, Danial Lashgari, Mia Pras-Raves, Eric J.M. Wever, Rodrigo García-Valiente, Adrie D. Dane, Perry D. Moerland (2024) ENCORE. A practical implementation to improve reproducibility and transparency of computational research. (Nature Communications, accepted for publication).

2. Van Kampen AHC, Mahamune U, Jongejan, A (2023) The standardized file system structure (FSS) navigator. Zenodo. DOI: https://doi.org/10.5281/zenodo.7985655.

3. Mahamune U, Moerland, PD, ……. ,Van Kampen AHC (2024) ENCORE. A case study in spatial transcriptomics. *In prep.*

4. van Kampen AHC, Jongejan A, van Schaik, BDC, ……., Holtrop AF, Wever, EJM, Dane, AD, Moerland, PD (2024) ENCORE. Improve reproducibility and transparency of data analysis in support project. I*n prep.*

ENCORE: A practical implementation to improve reproducibility and transparency of computational research

Supplementary Information

Description of predefined files and sub-directories

The standardized File System Structure (sFSS) / Project Compendium

To organize a computational project into a project compendium, ENCORE has defined an sFSS which is a predefined but flexible directory structure to organize conceptual information, data, code, documentation, and (intermediate) results (e.g., tables, figures, text files). Each directory of the sFSS contains one or more predefined files to provide explanation and guidance for documentation. The sFSS is also referred to as a **Project Compendium**.

Project Team and Compendium Recipient.

The (leader of the) **Project Team** is responsible for the organization and documentation of the project. The Step-by-Step ENCORE Guide and the explanation/instructions in the various README and other files are for the Project Team.

The **Compendium Recipient** is, for example, a direct colleague, a peer, or a reviewer who wants to inspect and/or reproduce the results of a project. Or it can be an archive such as Zenodo.

Pre-defined sub-directories and files in the standardized file system structure (sFSS)

The README Markdown files in the sFSS describe the content of the sub-directories and the content of the pre-defined files (including the README files) in more detail. The list below provides a summary of all sub-directories and pre-defined files.

Supplementary Figure 1. The standardized File System Structure (sFSS) and associated pre-defined files. Standardized directory structure of the sFSS containing pre-defined files (gold), which include README files (in Markdown format) that provide a documentation template and instructions. Note that the pre-defined files in the 'Data' directories (orange) and the '0_SoftwareEnvironment' subdirectories are only shown once. The names of the directories 'NameOfDataset_1' and 'NameOfComputation_1' are placeholders and should be replaced with more descriptive names. These directories can be replicated if multiple datasets are used or if different computation procedures are performed. Subdirectories shown in light blue are under version control using Git/GitHub. The '0' prefix ensures that the corresponding files/directories are always on top of the file list when using lexicographic ordering. The README.md in 'Processing' is the default GitHub repository README file and therefore does not have the '0' prefix.

Description of the sFSS sub-directories

\Manuscript

This directory contains the (draft) manuscript(s) corresponding to this computational project including figures, tables, and supplementary information.

\Data

\Data\NameOfDataset_1

\Data\NameOfDataset_1\Raw

\Data\NameOfDataset_1\Processed

\Data\NameOfDataset_1\Meta

These directories contain the raw, processed, and meta-data. Raw data comprises unprocessed data that come from the physical measurement device. Processed data comprises data obtained from collaborators or public databases and, consequently, not produced as part of the computational analyses. If the data (pre)processing is part of the computational analyses then, preferably, it should be placed in the \Results directory within \Processing. However, ENCORE allows the flexibility to store one's own processed data in the \Data\ NameOfDataset_1\Processed directory. Meta-data is the description of the data including the data license, description of the samples, experimental design, content and format of the data files, etc.

\Processing

Contains all sub-directories and pre-defined files related to the computational part of the project.

\Processing\0_SoftwareEnvironment

\Processing\0_SoftwareEnvironment\Anaconda

\Processing\0_SoftwareEnvironment\C++

\Processing\0_SoftwareEnvironment\Matlab

\Processing\0_SoftwareEnvironment\Python

\Processing\0_SoftwareEnvironment\R

One challenge that is only partially addressed by ENCORE concerns the preservation of the full computing environment. This environment is defined by (interdependencies of) the operating system, software tools, versions and dependencies, programming language libraries, etc. Gruning and co-workers proposed a software stack of interconnected technologies to preserve the computing environment (Gruning, Chilton, et al., 2018). This stack comprises (Bio)Conda (Anaconda Software Distribution, 2020; Gruning, Dale, et al., 2018) to provide virtual execution environments addressing software versions and dependencies, container platforms such as Docker (Nust et al., 2020) to preserve other aspects of the runtime environment, and virtual machines using cloud systems or dedicated applications such as VMware, to overcome the dependencies on the operating system and hardware. We are currently investigating how to best approach this within the ENCORE environment.

However, some basic information about the computing environment (e.g., export of Conda environments) can be stored in this directory.

The sub-directories provide basic information about different environments (e.g., R/RStudio, Python/PyCharm, Anaconda) for peers not familiar with the used computing environment. In addition, one may find other files such as cheat sheets, tutorials, and exports of (Anaconda) environments.

\Processing\Data
See \Data

\Processing \NameOfComputation_1

- This directory should also contain a conceptual description of applied methodology to improve transparency. For example,
    - Brief description of methods (and version) used including specification of the mathematical/statistical model, parameters, variables, references, etc.
    - If a new method is developed, this method should be described in full detail.
    - Description of why the selected or developed computational approach is valid for the research question that is addressed. This enables peers to make their own judgement about the approach and results.
    - Considered alternatives?
    - Detailed description of all data filtering, reduction, normalization etc. steps that are performed prior to the downstream analysis.
    - Avenues of exploration examined throughout development, including information about negative findings.

\Processing \NameOfComputation_1\Code
Contains the (in-house developed) software used for the computational analysis.

\Processing \NameOfComputation_1\CodeDocumentation
External (user) documentation of the code. Possibly automatically generated with documentation tools such as Sphinx.

\Processing \NameOfComputation_1\Data
See \Data

\Processing \NameOfComputation_1\NoteBooks
Notebooks ((web-based) interactive computing platform that combines live code, equations, narrative text, visualizations etc.) should be placed in this sub-directory.

\Processing \NameOfComputation_1\Results
(Intermediate) results (e.g., figures, tables) from the computational analysis. Record of intermediate results (preferably in a standardized format). It is strongly advised to generate hierarchical analysis output, allowing layers of increasing detail to be inspected. This can reveal discrepancies toward what is assumed and can in this way uncover bugs or faulty interpretation that is not apparent in the final results. It also allows any inconsistency to be tracked to the step where the problem occurs. Furthermore, it enables critical examination of the full process behind a result. It is

also advised to clearly document the intermediate/final results and the imposed hierarchy.

For any figure or table that ends up in a publication, report, or presentation at a meeting, the underlying data and a stand-alone piece of code should be available to regenerate the figure. This also allows easy modification of a figure and retrieval of the data corresponding to a figure (instead of having to redo a complete analysis). Equally important, the data corresponding to a figure can be further analyzed or inspected.

\Processing \NameOfComputation_1\Settings
This file/sub-directory concerns settings/parameters for the algorithms that have been developed. For settings related to the computing environment, see \0_SoftwareEnvironment for further instructions.

\ProjectDocumentation
This subdirectory contains (background) information about any part of the project. However, as a rule, documentation should be close to the component (e.g., data, code) that is described. For example, the documentation about the data should be in the \Data directory. However, more general information can be placed in the sub-directories in \ProjectDocumentation.

\ProjectDocumentation\BackgroundDocumentation
Documents relevant as project background documentation. For example, the project proposal, presentations from collaborators or peers (thus, not from the project team), and relevant tutorials about applied methodology.

\ProjectDocumentation\Literature
This subdirectory should contain relevant scientific literature (e.g., PDF files) obtained from PubMed, bioRxiv, etc. The PDF files should follow the naming convention Author-Year-Journal (or similar) to allow easy retrieval during project discussions. In addition, it should contain the reference manager (e.g., EndNote, Mendeley, BibTeX) file and an export to the standardized RIS format. The README file in this sub-directory should briefly describe the relevance of each paper (e.g., specific information relevant for the computational analyses).

\ProjectDocumentation\MyPresentations.
This sub-directory contains oral or poster presentations (and abstracts) given by the project team during meetings (e.g., at progress meetings, seminars, conferences).

\Sharing
Rationale. In general, the complete file system structure (FSS) and its contents should be shared unmodified with peers that aim at reproducing the computational analysis. However, in specific cases it might be desirable to share only parts of the sFSS and/or restructure the sFSS. The reduced and/or restructured sFSS is then stored (as a compressed file) in the \Sharing directory. This file should at least indicate what one did (not) share and how/why the sFSS was restructured. In addition, document when and with whom this directory was shared.

Typical use. A typical use of the \Sharing directory is for support projects in which computational analyses are performed for other researchers as a service (e.g., biomedical, clinical researchers). These researchers might only be interested in the final results (figures and tables) and not in the code that produced these results. That is, they will not aim to repeat the analyses. In such a situation, the results and tables can be shared in a (flat) structure that is more convenient for them to browse and use, and that leaves out all code and background documentation.

Restrictions. Sharing an sFSS with an (external) colleague may be restricted due to, for example, copyright on PDF files of papers, sensitive/private information (in LabJournal.docx), non-open-access of data, etc. Make sure to remove such information from \Sharing.

Description of the sFSS pre-defined files

Sharing the project with a Compendium Recipient

- Several files in the sFSS are provided in different formats (e.g., text, Markdown, LaTex, Microsoft Word, HTML). The Project Team has to remove the formats that will not be used. This prevents that these redundant files cause confusion when the project compendium is archived or shared with a Compendium Recipient.
- The Step-by-Step-ENCORE-Guide can be removed since this document is only relevant for a Project Team setting up and using ENCORE.
- Test_Navigate_Module.py can be removed.

0_README.md / README.md

Throughout the sFSS there are README files that explain the content of the sub-directories and provide instructions and a template to guide documentation of the project. Most of these files are so-called Markdown files that can be opened in any text editor, but require a Markdown viewer (e.g., Notepad++, Typora) to show the resulting document.

To facilitate first-time users, the README file in the root directory also serves as the landing page of the ENCORE template GitHub repository.

Each *_README file starts with a '0_' prefix to ensure it appears at the top of the file list. The only exception is the README.md file in the /Processing directory, which is the default GitHub README file that should not contain a prefix. Because the sFSS (and not GitHub) is the entry point for a project, the GitHub README.md file does not necessarily have to contain a project description. However, you may want to copy the information from 0_PROJECT.md (see below) into this README.md file. More importantly, it should provide an explanation of the code in the Processing directory and instructions about its execution.

0_PROJECT.md

Short description of project, contact person, and project team. This file is used by the sFSS Navigator.

0_GETTINGSTARTED.txt.

Template document (plain text format). One can copy this file to one's favorite editor to add content. Examples:

- **0_GETTINGSTARTED.docx.** Template document (Microsoft Word format containing an example of how to include links). The DOCX file can be saved as HTML (make sure to use UTF-8 encoding).
- **0_GETTINGSTARTED.tex.** Template document (LaTeX format containing an example of how to include links). The LaTeX file can be converted to HTML with pandoc (https://pandoc.org/index.html).
- **0_GETTINGSTARTED.html.** Example of exported HTML file used by the sFSS Navigator.

The main use of the GETTINGSTARTED files is to guide a first-time user of a finished project to the most important aspects (e.g., results, code) of the project before he/she manually explores other? information contained in the sFSS. The GETTINGSTARTED files provide links to the relevant sub-directories and files. 'Getting started' templates are provided in different file formats, which can be converted to HTML once finished. The 0_PROJECT.md and 0_GETTINGSTARTED.html files are used by the sFSS Navigator

Help files

- **1_Step-by-Step-ENCORE-Guide.{pdf, docx}.**
  User guide for ENCORE, describing the standardized File System Structure (sFSS) and how to set up a corresponding GitHub repository.

General

- **2_CITATION.md.** How to cite ENCORE and the sFSS Navigator.
- **3_LICENSE.md.** LICENSE for the ENCORE template.
- **.FSSignore.** Currently not used but to be used with an application that selects all files needed for sharing.

sFSS Navigator

- **Navigate.py.** Standalone Python 3 script to generate Navigate.html used for navigating the sFSS. Can be run from the command line (Navigate.py -h)
- **Navigate_U.sh.** Shell script to run Navigate on Unix/Linux systems. Change the first line (#!/usr/bin/Python) if necessary. Make executable using chmod +x
- **Navigate.html.** Open in a browser to navigate the standardized file system. This file is created after running Navigate.py.

There are also executables available for Windows and macOS. These are available from the latest GitHub release and from Zenodo (DOI: https://doi.org/10.5281/zenodo.7985655):

- **Navigate_W.exe.** Windows executable if one doesn't have Python installed (Navigate.exe -h).
- **Navigate_M.** MacOS executable (macOS 13.3.1 (Ventura), Apple M1)
- **Navigate_MacIntel.** MacOS executable (macOS 10.13.6 (High Sierra), Intel Core i5)

Test_Navigate_Module.py.
Python script to show how to use Navigate.py as a module in other Python scripts. This may help to keep Navigate.html up to date without manually executing Navigate.py.

Navigation.conf.
Configuration file for the sFSS Navigator.

\ProjectDocumentation\LabJournal.{docx, md, tex, txt}.
Templates in different file formats for the lab journal.

In general, any documentation should be kept in the sub-directory that it describes. Thus, use the 0_README.md and/or additional (e.g., PowerPoint) files to document the data, software, and results in their respective directories.

The lab journal should contain more general documentation. For example,

- General information and concepts
- Summaries of project discussions
- Steps to be taken
- New (future) research ideas
- Pointers to the location of certain pieces of information

In addition, it may also contain integrated parts of the various README files whenever useful (but keep it consistent with the source files). Include figures and tables when necessary.

Although, strictly speaking, a lab journal is not intended for recording new/future ideas or providing summaries of discussions, it is important for a research group to also have a record of this. Therefore, these can also be included in the lab journal.

Parts of the lab journal that should not be shared with peers (e.g., new research ideas) should be clearly labelled with 'Not for sharing' such that one can easily remove these parts. Alternatively, one may maintain two separate documents.

If necessary, ENCORE allows to maintain lab journals in multiple sub-directories. For example, one may decide to have a separate lab journal in '\Processing\NameOfComputation_1' for a specific part of the computational analysis.

\Processing\github.txt.
Provides the URL to the corresponding GitHub repository and any other relevant information about the repository. This file is used by the sFSS Navigator.

\Processing\gitignore-FSS-template.txt and .gitignore
The template file should be adapted (if needed) and then be renamed to .gitignore. It contains instructions for Git about files that should not be synchronized with the GitHub repository (e.g., data and results). This file should be modified depending on the contents of \Processing. Optionally, one can use other language-specific templates that are also found in \Processing*:*

- gitignore-C++-template.txt
- gitignore-FSS-template.txt
- gitignore-JetBrains-template.txt
- gitignore-Matlab-template.txt
- gitignore-Python-template.txt
- gitignore-R-template.txt

To use any of these templates, simply merge its content into .gitignore. It is considered good practice to keep a single .gitignore in the top-level directory and not in individual subdirectories, which would make debugging more troublesome.

ENCORE: A practical implementation to improve reproducibility and transparency of computational research

Supplementary Information

Example of a 0_README.md file.

This specific file is copied from
\ENCORE\Processing\NameOfComputation_1\Code

Most README files contain three sections to guide the researcher in providing relevant and sufficient documentation. The information that is asked for is not exhaustive and additional information might need to be provided for specific projects.

The 'Explanation' section provides general information about the type of documentation that should be provided and the reason why. The 'Instruction' section gives more specific instructions to the researcher. Finally, the file ends with a 'Template' that asks the researcher to specify a minimum amount of information. Not all requested information may always be relevant. Part of the requested information may also reside in separate documents, if that is more convenient.

The explanation and instructions are not exhaustive. Much more can, for example, be said about best-practices for code development. However, this provides a starting point for the researcher without being overwhelmed. In the future, we aim to add references to relevant resources (which are now partially in the main text).

CODE

Explanation:

This directory contains the code you use (and developed) as part of this project.

Software Engineering

Over the past two decades, an increasing number of researchers have become involved in computational research. Many of these researchers have never received formal training in software engineering. Software engineering is a discipline in its own and includes the design, implementation, documentation, testing and deployment of software.

The lack of good software engineering practices can negatively affect the reliability and transparency of software, and consequently, its transparency and reproducibility. Poorly designed and documented software can be difficult for peers to understand, use, modify, and debug.

Additionally, inadequate software engineering practices can lead to a situation where there is no way to verify if the code used to generate computational results is functioning as the researcher intends.

It is important to recognize that there is a vast field dedicated to software engineering, encompassing a wide range of tools and best practices that can be utilized to enhance software development.

Below we very briefly provide a few pointers to improve your software engineering practices.

Coding

Following best practices for scripting, functional programming, or objective-oriented programming may significantly improve the quality of the code but requires training and experience.

Clean code

Write programs for people, not computers (code should be easily read and understood). Adopting clean code practices helps to standardize and organize software code in order to enhance readability. This allows developers to concentrate on core functionality and reduce errors. Readability helps reproducibility and transparency.

A few general rules:

- *Reduce complexity. Decompose programs into functions and modules (instead of making very long scripts).*
- *Limit number of function arguments and length of functions.*
- *Be ruthless about eliminating duplication (use functions).*
- *Always search for well-maintained software libraries that do what you need, and test these libraries before relying on them.*
- *Do not comment and uncomment sections of code to control a program's behaviour. This is a recipe for making your software irreproducible.*
- *Choose a style guide—And stick with it (see for example, https://peps.python.org/pep-0008/, https://style.tidyverse.org/, https://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html)*
- *Give functions and variables meaningful names.*
- *Make dependencies and requirements explicit*

- *Refactor as needed (process of restructuring your code without changing its interface).*
- *Write error messages that provide solutions or point to your documentation.*

Code versioning

ENCORE is based on Git/GitHub for versioning of code and code documentation. It may be worthwhile to dive into the Git/GitHub world at greater detail. For example:

- *Understanding the Staging Area*
- *Git branching strategies*
- *Perez-Riverol, Y., Gatto, L., Wang, R., Sachsenberg, T., Uszkoreit, J., Leprevost Fda, V., Fufezan, C., Ternent, T., Eglen, S. J., Katz, D. S., Pollard, T. J., Konovalov, A., Flight, R. M., Blin, K., & Vizcaino, J. A. (2016). Ten Simple Rules for Taking Advantage of Git and GitHub. PLoS Comput Biol, 12(7), e1004947.*
- *Comprehensive Guide to Version Control*

Code testing

Interrogate specific and isolated coding behaviour to reduce coding errors and ensure intended functionality, especially as code increases in complexity. Describe if software tests have been performed and how to re-run these tests.

From IBM: There are many different types of software tests, each with specific objectives and strategies:

- ***Acceptance testing:** Verifying whether the whole system works as intended.*
- ***Code review:** Confirming that new and modified software is following an organization's coding standards and adheres to its best practices.*
- ***Integration testing:** Ensuring that software components or functions operate together.*
- ***Unit testing:** Validating that each software unit runs as expected. A unit is the smallest testable component of an application.*
- ***Functional testing:** Checking functions by emulating business scenarios, based on functional requirements.*
- ***Performance testing:** Testing how the software runs under different workloads. Load testing, for example, is used to evaluate performance under real-life load conditions.*
- ***Regression testing:** Checking whether new features break or degrade functionality. Sanity testing can be used to verify menus, functions and commands at the surface level, when there is no time for a full regression test.*
- ***Security testing:** Validating that your software is not open to hackers or other malicious types of vulnerabilities that might be exploited to deny access to your services or cause them to perform incorrectly.*
- ***Stress testing:** Testing how much strain the system can take before it fails. Stress testing is considered to be a type of non-functional testing.*

- ***Usability testing:*** *Validating how well a customer can use a system or web application to complete a task.*

Not all of these tests are equally important when it comes to scientific software.

Code documentation

Write comments as you code (not afterwards). Modern IDEs can assist in to automatically generate documentation strings as you write code, which removes the burden of having to remember to write comments. (e.g., PyCharm, DataSpell from JetBrains).

Be aware of guidelines and tools to (automatically) generate documentation such as Sphinx using Python docstrings, and r2readthedocs, and roxygen2 for R, will also help to improve reproducibility. We used Sphinx for the documentation of the sFSS Navigator.

There are different types of documentation:

- *Requirements Specification*
- *Software Design*
- ***(External) source code documentation***
  - *Place a brief explanatory comment at the start of every program.*
  - *Document the input and output of your script/functions.*
  - *Describe what the code is doing and why.*
- *Testing Requirements*
- ***End-User Instructions (including a quick-start guide)***
  - *How to install and configure the software.*
  - *Where to find its full documentation.*
  - *Examples of how to execute the code to produce certain output. Provide a simple example or test dataset.*
  - *Under what license it's released.*
- *Include a help command for command line interfaces*

In bold the documentation that should be provided at a minimum in the context of ENCORE.

Integrated development environment *(IDE)*

Integrated Development Environments help to improve software. Consider using an IDE, e.g., Visual Studio Code, PyCharm and DataSpell from JetBrains (https://www.jetbrains.com/), RStudio (https://www.rstudio.com/).

IDEs offer a various advantages:

- *Build in compilation and build tools*
- *Code editing features (e.g., syntax highlighting, code completion, code refactoring)*
- *Code debugging and testing*

- *Code documentation*
- *Integration with versioning systems*
- *Integration with AI-based tools such as Copilot*
- *Extensibility (plugins) and Customization (workflow)*

AI-based tools

Large Language Models (LLMs) increasingly play a role in software development, testing, and documentation (e.g., Copilot, ChatGPT). Copilot is also integrated with GitHub Codespaces. It is worthwhile to try out these new tools.

Preserve your computing environment

See 0_README.md in 0_SoftwareEnvironment

Instructions:

- *Remove all Instructions and the Explanation once you have completed the template.*
- *Level of detail: Information provided should be sufficient for someone who was not involved in the project and/or has limited knowledge about the topic, to understand and reproduce the project.*

Software documentation

Ensure that you have properly documented your code. Not all types of software documentation on this list need to be written for a single research or support project but a combination of several should be selected depending on the nature of the project. Whether more or less documentation is needed for your project will depend on its scale and complexity. For a research project you need at least to write (external) source code documentation and user documentation.

====== TEMPLATE STARTS HERE ======

What (external) documentation is available?

Documentation files: [filename; short description of content]

How did you test the code:

Script 1: [description]

Script 2: [description]

Description of manual steps:

Relation between the scripts, and user instructions for execution:

ENCORE: A practical implementation to improve reproducibility and transparency of computational research

Supplementary Information

The use of complementary (GitHub) tools in combination with ENCORE

This supplementary file discusses the use of complementary software tools alongside ENCORE to improve computational reproducibility.

Contents

1 ENCORE requirements

In the main text we explained that ENCORE was driven by eight main requirements. Our view on the use of complementary (GitHub) tools with ENCORE mainly concerns requirements 1, 5, 7 and 8:

1. **Consist of a single self-contained project compendium**. The computational project should be organized and available as a self-contained and integrated compendium of data, code, results, and (conceptual) documentation, stored at a single location. It should also be easily transferable to other researchers or reviewers without breaking its internal consistency.
2. **Facilitate transparency and documentation**. ENCORE should facilitate transparency and a deep understanding (e.g., addressing why specific methods were selected and how these were applied) of the project through its standardized structure and documentation of concepts, methodology, data, code, and results.
3. **Enable reproducibility**. The project compendium should enable an external researcher to autonomously execute and understand the computational techniques and recreate the (published) outcomes.
4. **Adhere to proposed guidelines**. ENCORE should follow published guidelines for computational reproducibility as much as possible.
5. **Enable version control**. ENCORE should allow version control of code and code documentation.
6. **Facilitate harmonization**. The ENCORE approach itself should be standardized and well-documented such that it can easily be adopted by any researcher. This allows harmonization within research groups, enabling further joint development of best practices within the ENCORE framework. Moreover, harmonization also facilitates checking transparency and reproducibility prior to publication by direct colleagues.
7. **Provide a generic approach.** ENCORE should be agnostic to the type of computational project (e.g., statistical analysis, mathematical modelling), data, programming language, and ICT infrastructure (e.g., operating system and computer hardware). ENCORE should make use of a software versioning system but otherwise should not rely on tools for project management, data processing, etc.
8. **Allow adaptation to different styles of working**. ENCORE should leave sufficient flexibility to accommodate different styles of working. The underlying sFSS should be accessible from any software tool the researcher might be using.

2 Requirement 5. Software version control

From the initial stages of ENCORE development, we recognized the need for a software versioning system for which we selected Git/GitHub. Consequently, the ENCORE template (documentation and github.txt in \Processing) and Step-by-Step Guide are based on the use of GitHub. However, any other version control platform such as GitLab, BitBucket, Subversion, or Mercurial could be used, as this would not alter the ENCORE approach.

3 Requirement 8. Allow adaptation to different styles of working

Most researchers develop their own methods for organizing research projects and utilize specific software tools for software development, computation, project management, reporting, etc. Therefore, any platform aimed at supporting reproducibility must accommodate diverse working

styles and be compatible with the software tools researchers already use. Imposing an entirely different way of working (e.g., different tools) is, in our view, too disruptive and a recipe for failure. Although ENCORE prescribes a certain project organization structure, we believe it also provides sufficient flexibility. Additionally, since ENCORE is file system-based, it will be compatible with most software used by researchers.

4 Requirement 1. Single self-contained project compendium & Requirement 7. Provide a generic approach.

We strongly believe that for a platform like ENCORE to be successful, it should neither depend on specific software tools for project management, data storage, or processing, nor on particular hardware or operating systems. Such dependencies would be too restrictive and disruptive for many researchers, resulting in a platform that will not be widely adopted.

Currently, ENCORE only requires the use of a software versioning system (e.g., GitHub or an alternative platform) and a small Python program (sFSS Navigator). However, this Python program is also available as an executable for Windows and macOS, and as a shell script for Unix, ensuring that it can be used even if Python is not installed on Windows or macOS. Moreover, the sFSS Navigator is not essential for working with ENCORE and is only meant to generate an HTML file (Navigate.html) for the compendium recipient.

4.1 'Neglecting' software tools: Good or bad?

Many tools that significantly enhance computational reproducibility may initially appear to be overlooked by ENCORE. Examples mentioned in the main text include software versioning systems (e.g., GitHub, BitBucket, GitLab), tools to preserve the computing environment (e.g., conda, renv, Docker, Apptainer (formely Singularity)), workflow management systems (e.g., Snakemake, NextFlow, Galaxy, Knime), Integrated Development Environments (e.g., Visual Studio Code, NetBeans, PyCharm), software documentation tools (e.g., Doxygen, Sphinx), AI-based tools to support software engineering (e.g., Copilot, ChatGPT, Cody AI), and project management and documentation tools (e.g., Wikis, Trello, Word, PowerPoint, LaTeX). Additionally, platforms like GitHub, GitPod, and AWS Cloud offer cloud-based Integrated Development Environments (IDEs) to provide a range of functionalities for software development. For example, GitHub provides an integrated toolset that includes software versioning, wikis, project discussions, integration with Copilot, workflows (i.e., GitHub Action) and a development environment (Codespaces/containers). The use of such platform seems attractive since it may largely contribute to reproducibility. However, the choice of tools depends greatly on the researcher's preferences and the specific requirements of a project.

By design, ENCORE does not impose the use of any specific tools other than basic Git/GitHub functionalities for software versioning. This flexibility does not preclude the use of additional tools, some of which are essential for achieving reproducibility, such as environment management and containerization. We emphasize that ENCORE is neither intended to replace these tools, nor does it exclude their use. In fact, we believe that aforementioned tools should be utilized as much as possible in combination with ENCORE. However, ENCORE leaves it to the researcher to decide which tools to use. Incorporating specific tools in ENCORE by design would make the approach far less attractive to

the broader community given individual preferences, expertise, and experiences.

4.2 An exception: approaches for the preservation of software environment

The preservation of the software environment using tools like conda, renv, Docker, Apptainer, and Virtual Machines, is essential for reproducibility. We are currently investigating within our group how to best include such tools in the ENCORE specifications. Likely, in a next ENCORE version we will include a set of preferred tools and approaches to preserve the computing environment. We expect to do this in a similar way we do now for Git/GitHub, i.e., provide detailed instructions and automation scripts.

4.3 Can GitHub functionalities provide an alternative for ENCORE?

Since ENCORE relies on Git/GitHub for software versioning, it is worthwhile to evaluate the extent to which we can leverage other functionalities of GitHub, or if the GitHub platform could serve as an alternative to ENCORE.

The ENCORE file system structure (sFSS) is the central hub and entry point for a project (**Figure 1**). While ENCORE incorporates basic Git/GitHub functionalities for software versioning, it is not primarily based on Git/GitHub. ENCORE synchronizes only part of a project with the GitHub repository, specifically a selection of files (code, notebooks, and code documentation) within the \Processing directory. The ENCORE sFSS is intended to be shared, whereas sharing the GitHub repository is optional and only necessary if the recipient is interested in accessing previous software versions.

We believe it is important to keep in mind that GitHub is designed to support collaborative software development, with versioning of code and text-based documents as core functionalities. GitHub is not designed to support (large-scale) data analysis projects although additional GitHub functionalities (described below) make it a more general computation platform. Let us first review part of the GitHub functionalities in the context of ENCORE.

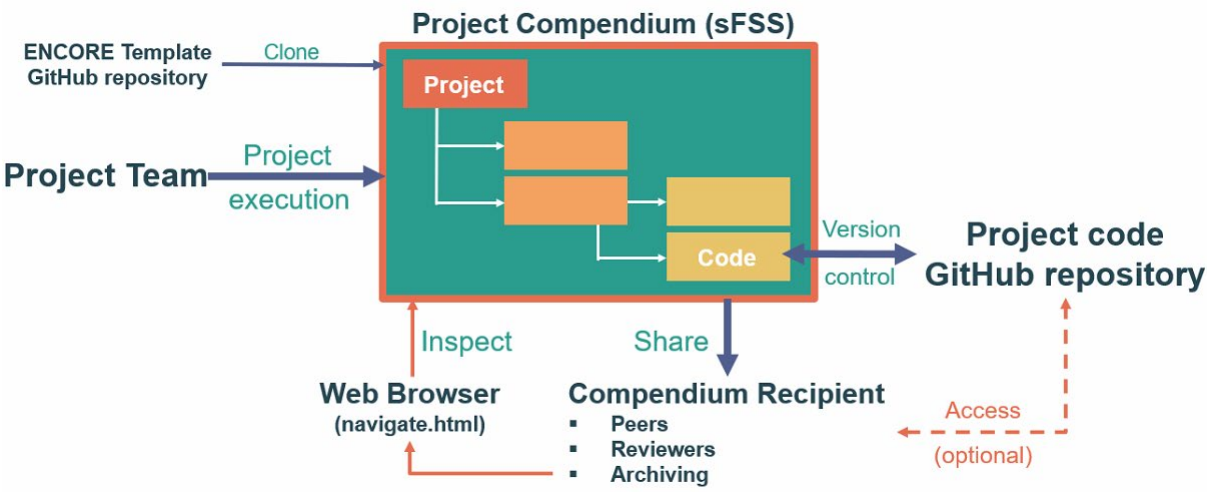


Figure 1. The sFSS and its environment. The green box denotes the Project Compendium (sFSS) with part of the directory structure shown. The sFSS is the central point of entry for a project and is initially cloned from the ENCORE template GitHub repository when starting a new project. The project team is responsible for the organization and documentation of the project. Only the code and code documentation within the project compendium are synchronized to a project specific GitHub repository. An sFSS project compendium can be shared with a compendium recipient. The compendium recipient starts exploring the project by opening navigate.html in a web browser. (copied from main text).

4.3.1 GitHub Large File Storage (LFS)

GitHub LFS (Large File Storage) is an extension to Git that enables handling large (data) files efficiently. Traditional Git is not optimized for storing large (binary) files which can cause performance issues and increase the repository size too much. GitHub LFS addresses this problem by storing large files outside the main Git repository on a separate server and only keeping pointers to these large files in the repository itself. Only versions of the large files needed for the current 'checkout' are downloaded.

Currently, every account using Git LFS receives 1 Gbyte of free storage and 1 Gbyte/month of free bandwidth, which is by far not sufficient for most projects performed in our group. To extend one needs to acquire additional data packs. One data pack cost $5/month and provides a monthly quota of 50 Gbyte for bandwidth (that is also consumed by anyone cloning the repository) and 50 Gbyte for storage. You can purchase as many data packs as you need. For example, if you need 150 Gbyte of storage, you'd need three data packs, which can quickly become expensive, especially for long-term data retention. Additionally, data stored in the local (harddisk) Git repository also consumes (paid) local storage and creates additional costs.

- **Example.** For example, for AIRR-seq B-cell/T-cell repertoire experiments we would need about 150 Gbytes for a full analysis corresponding to 3 data packages ($15 / month = $180 /year). Per year we analyze multiple projects with total data storage of about 800 GB per year ($960 /year) with currently about 8000 Gbyte of data ($9600 /year) that we currently store for free on a Dutch cloud-storage facility.

In principle, LFS allows to host complete ENCORE projects on GitHub, but this seems only useful if one intends to use this data with Codespaces, since otherwise the data can be shared/archived as part of

the ENCORE sFSS. The pricing of LFS may also be a significant limitation, potentially increasing storage costs if free storage is not available, as one would likely maintain a local copy as well. Additionally, storing confidential or patient data outside the firewall of a research institute or hospital may not be permitted, even temporarily. Furthermore, the fact that GitHub is owned by Microsoft could raise additional concerns regarding data privacy and security.

Documentation: https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-git-large-file-storage

4.3.2 GitHub Codespaces/ development containers

Codespaces is a very useful functionality if one needs a cloud-based IDE and if every member of the project team must use the same development environment and tools. Within Codespaces, a large range of extensions and tools can be installed. When working with Codespaces, one is using a Docker container on a virtual machine, which largely supports reproducibility. The Docker image is in the Cloud to be used by anyone. However, Codespaces are meant for software development, not necessarily for (large scale) data analysis/simulations. Occasionally, Codespaces may be too restrictive if one needs specific hardware (e.g., GPUs) or complete control over the environment. Nevertheless, Codespaces may prove very useful for software development and testing, but also for specific types of data analysis projects.

In principle, one could host a complete ENCORE project (i.e., not only the \Processing directory; **Figure 1**) on GitHub and make use of Codespaces for the software development and data analysis/simulations/etc. This would improve reproducibility but in practice one would run into various limitations:

- **GitHub file size and storage limits**. ENCORE contains much more than only code and documentation. Given current GitHub limits (<5 Gbyte for a repository, and 100 Mbyte for single files) it would in many cases not be possible to host a complete project on GitHub.

- **Large datasets**. Projects that make use of large datasets are limited by the default GitHub storage limits and/or Codespaces limits (currently, 15 GB/month and 120 core hours/month for a GitHub Free personal account; https://docs.github.com/en/billing/managing-billing-for-github-codespaces/about-billing-for-github-codespaces#pricing-for-paid-usage). Consequently, for large datasets one needs to resort to, for example, GitHub LFS (see below), which also has limits for free accounts. Moreover, using Codespaces in combination with LFS implies two separate bills (one for storage used in LFS and one for storage in Codespaces). Also note that a computational project may produce a large number of (large output) files (data tables, images, objects, etc.) that may further increase storage requirements and costs, and requires some mechanism to store these output files on LFS if/once committed to the repository (unclear if one can push new output files to LFS from Codespaces at all).

    - **Example.** For (single-cell) RNA-seq transcriptomics experiments we receive data files of up to 15 Gbytes per sample. For lipidomics/metabolomics studies we typically have between 120 and 500 Mbyte per sample. For studies with multiple samples (tens to hundreds) for which many and/or large (intermediate) output files are generated, the

storage requirement will rapidly increase. For example, for AIRR-seq experiments we typically receive about 15 Gbytes of raw data for a single project consisting of multiple samples, which is increased to about 150 Gbytes after the full analyses.

- **Computation time**. For the free version of Codespaces computation time is limited to 120 core hours/month, which might be sufficient to test (parts of) software but in general is not sufficient for a full data analysis/simulation. Nowadays, many researchers run their analyses/simulations on local/national computer facilities, which may offer more compute power and/or other functionalities than those provided by GitHub Codespaces, and/or may be free, cheaper, or more powerful.

    - **Example.** For a typical AIRR-seq project we require about 1000 CPU hours. On a yearly basis we use about 50,000 CPU hours for AIRR-seq projects. We currently run these projects on the Dutch national compute infrastructure (https://www.surf.nl/en/services) for free. Using a Codespaces machine with 32 cores, we would pay $4500 yearly. In addition, we would have to adapt our code for using a multicore machine since currently our software is designed for multiple single-core CPUs. If we would use parallel computing using multiple single-core CPUs with Codespaces then the costs are about twice as high, that is $9000 yearly. Moreover, we would run into data size/file limits in this case.

In summary, for projects that don't face CodeSpaces CPU or storage limits, Codespaces could be a great functionality for the development and testing of the software and small-scale data analyses, while at the same time preserving the compute environment. This would be in full agreement with ENCORE requirements since the GitHub repository (including the container specifications) could still be synchronized with the local ENCORE copy on one's harddisk. Codespaces also integrates with local IDEs such as DataSpell/PyCharm (JetBrains) and Visual Studio Code. However, for larger data analysis/simulation projects one probably would run into GitHub/Codespaces CPU or storage limitations (even for paid usage). Moreover, researchers may prefer to use alternative compute infrastructures and/or alternative approaches towards software development.

Documentation: https://docs.github.com/en/codespaces/overview

4.3.3 GitHub Copilot.

Large Language Models (LLMs) such as those used by GitHub Copilot offer valuable functionalities for coding assistance, code documentation generation, and writing unit tests. GitHub 'Copilot Chat' is a chat interface to interact with GitHub Copilot within GitHub Codespaces or within supported IDEs (e.g., PyCharm/Dataspell from JetBrains). LLMs can also be used independently of the GitHub/Codespaces platform. Therefore, in general, LLMs can easily be used as part of the ENCORE framework. However, GitHub Copilot or alternatives like Codeium AlphaCode, ChatGPT Plus are not for free and a subscription is required. Pricing for Copilot is currently 100 USD/year for individual users, which is not overly expensive given the potential benefits.

Documentation: https://github.com/features/copilot, https://docs.github.com/en/copilot/github-copilot-chat

4.3.4 GitHub Projects

Projects is a basic cloud-based project management tool that is visible for all project members. One of its key advantages is that it can be linked to multiple repositories, but it does not become part of a specific repository. In the context of ENCORE, we are not in favor of using Projects, since it is not in agreement with the first ENCORE requirement (self-contained project compendium).

Documentation: https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects

4.3.5 GitHub Issues

Issues can be used to track ideas, bugs, etc. and can be used in the context of ENCORE but should not contain important documentation or discussions that we require to be part of ENCORE (first requirement: self-contained project compendium). Consequently, we mainly use it to document bugs or other minor software issues.

Documentation: https://github.com/features/issues

4.3.6 GitHub Discussions

GitHub Discussions is a platform to share information, ask questions, or have discussions. It is possible to attach documents to the discussion but is less suited for large files. A 'Discussion' is linked to a specific repository but is not part of the repository and, therefore, cannot be synchronized with the repository. Currently, the Discussions also cannot be exported. Consequently, it is not in agreement with the first ENCORE requirement (self-contained project compendium). However, Discussions has clear advantages. Project members can easily engage in project/software discussions that can be organized in different sections and threads. Discussions can be linked to GitHub Issues. However, we believe it is better to have all documentation within the sFSS and close to the files (e.g., data, results, etc) it is documenting, which is more difficult to achieve with a separate (cloud-based) forum. Currently, many of our ENCORE projects are hosted on the computers of the project members and synchronized to a cloud system (e.g., Dropbox, SURFdrive), making them accessible to all project members. While they can use the LabJournal.docx or other files to engage in discussions, this method is admittedly less convenient. However, a project team may decide to use a cloud-based system for project discussions. For example, something simple like Google would be useful and also allows to download Discussions to the ENCORE project at the moment of sharing or archiving.

However, we recently started to use 'Discussions' for the ENCORE template repository (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/discussions). This provides a channel to communicate with the scientific community about ENCORE. This Discussion section now reflects part of the steps that are to be addressed as part of ongoing discussions and development.

Documentation: https://docs.github.com/en/discussions

4.3.7 GitHub actions

GitHub Actions enable the setup and execution of software development workflows as part of a repository. This can, for example, be used to automatically build, test, and deploy software. Actions is

probably more suited for (multi-member) software development projects and less suited for initiating data analysis/simulations. In the context of ENCORE, Actions might be useful for automatically running unit tests when new code is pushed to the repository. Using Actions, these unit tests can be executed on a GitHub or local server (hosting the data). Depending on the setup, limits to the use of Actions may apply (https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration#usage-limits).

Documentation: https://docs.github.com/en/actions

4.3.8 GitHub Wiki

"A wiki is a form of online hypertext publication that is collaboratively edited and managed by its own audience directly through a web browser. A typical wiki contains multiple pages that can either be edited by the public or limited to use within an organization for maintaining its internal knowledge base" (https://en.wikipedia.org/wiki/Wiki). A GitHub Wiki is not part of the repository itself but can be synchronized separately but this will not pull/clone images used in the Wiki documents.

We have been using Wiki's in the past for project documentation. For ENCORE we did not consider the use of the GitHub Wiki because (i) we don't want to rely on a specific documentation system outside ENCORE (first requirement). Although, it is possible to pull/clone the Wiki associated with a repository, this will not clone any images that are used in the documentation. In addition, within ENCORE we prefer to have documentation in the subdirectory of relevance, (ii) Markdown formatting is still too limited in specific cases. (iii) Wiki's are not for free for our GitHub Organization account unless a repository is public which is neither always desired nor possible (e.g., confidential information) during the execution of a project.

However, we created a WIKI for the ENCORE template GitHub repository (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE/wiki) and for the ENCORE-AUTOMATION repository (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE_AUTOMATION/wiki) containing general basic information and documentation.

Documentation: https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis

5 Summary

This document describes our view on the use of complementary software tools alongside ENCORE to further improve reproducibility, and in the context of four primary requirements that guided the development of ENCORE. Specifically, we focused on requirements 1, 5, 7, and 8:

1. **Consist of a single self-contained project compendium**. The computational project should be organized and available as a self-contained and integrated compendium of data, code, results, and (conceptual) documentation, stored at a single location. It should also be easily transferable to other researchers or reviewers without breaking its internal consistency.
5. **Enable version control**. ENCORE should allow version control of code and code documentation.

7. **Provide a generic approach.** ENCORE should be agnostic to the type of computational project (e.g., statistical analysis, mathematical modelling), data, programming language, and ICT infrastructure (e.g., operating system and computer hardware). ENCORE should make use of a software versioning system but otherwise should not rely on tools for project management, data processing, etc.
8. **Allow adaptation to different styles of working**. ENCORE should leave sufficient flexibility to accommodate different styles of working. The underlying sFSS should be accessible from any software tool the researcher might be using.

With respect to the use of complementary tools our view is as follows:

1. Any platform designed to support reproducibility, including ENCORE, should avoid imposing significant restrictions on researchers (e.g., in terms of use of software tools). Excessive constraints may be perceived by individual researchers as too disruptive, leading to a platform that will not be adopted by the community.

2. ENCORE is designed to accommodate various styles of working and to be compatible with a large range of software tools. Consequently, it does not impose the use of any specific software tool except for Git/GitHub (**Figure 1**; which is easily replaced with another versioning system). However, ENCORE users are encouraged to utilize complementary tools that enhance reproducibility, including those for (i) preservation of the compute environment, (ii) software development, (iii) workflow management, (iv) (software) documentation, and (v) project management. In fact, some of these tools are essential for improving reproducibility but currently it is left to the individual researcher to make appropriate choices.

3. The first ENCORE requirement (a single self-contained project compendium) excludes the use of external of (cloud-based) tools/platforms to host project documentation, project discussions, data, etc but do not allow to synchronize/download this information into the ENCORE project. In addition, for project documentation we prefer to store documentation in the appropriate ENCORE subdirectories.

Since ENCORE relies on Git/GitHub for software versioning, it raises the question of how other functionalities of GitHub can complement ENCORE or whether the GitHub platform could serve as an alternative to ENCORE. We summarize our evaluation in the following points:

4. Git/GitHub has been developed for software development and version control of text-based documents, while the focus of ENCORE is on computational research which includes but goes beyond mere software development. ENCORE, therefore, includes much more than code, and using the GitHub platform as a replacement for ENCORE could be considered misuse of the GitHub platform.

5. GitHub offers numerous functionalities that could enhance reproducibility. However, there may be limitations depending on the type of project. For example, projects with high CPU or storage requirements may face functional constraints and incur additional costs. Additionally, not all GitHub functionalities align with the first ENCORE requirement of a single self-

contained project compendium.

6. A researcher may prefer the use of (free/cheaper) local/national compute infrastructure, or there might be data privacy and confidentiality issues that prevent the use of GitHub for large-scale calculations and/or data storage.

Therefore, it is up to individual researchers to decide which GitHub functionalities to use in conjunction with ENCORE for their specific projects.