ENCORE

Step by Step ENCORE Guide

ENhancing COmputational REproducibility

Date: 19 February 2024 **Version:** 13	**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 **FSS 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.

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. The main parts of the template are shown in *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 FSS 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).*

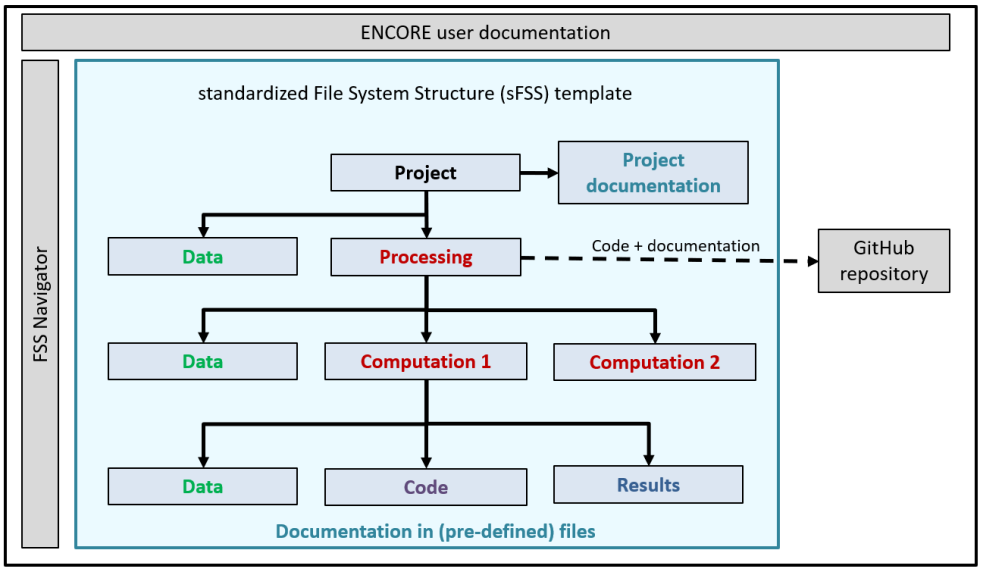

Figure 1. The main parts of the standardized File System Structure (sFSS) template. Project corresponds to the root directory of the template. Other blocks are sub-directories. Data can be organized at several levels depending of the preference for the project. Processing contains one or more computations (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?

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.

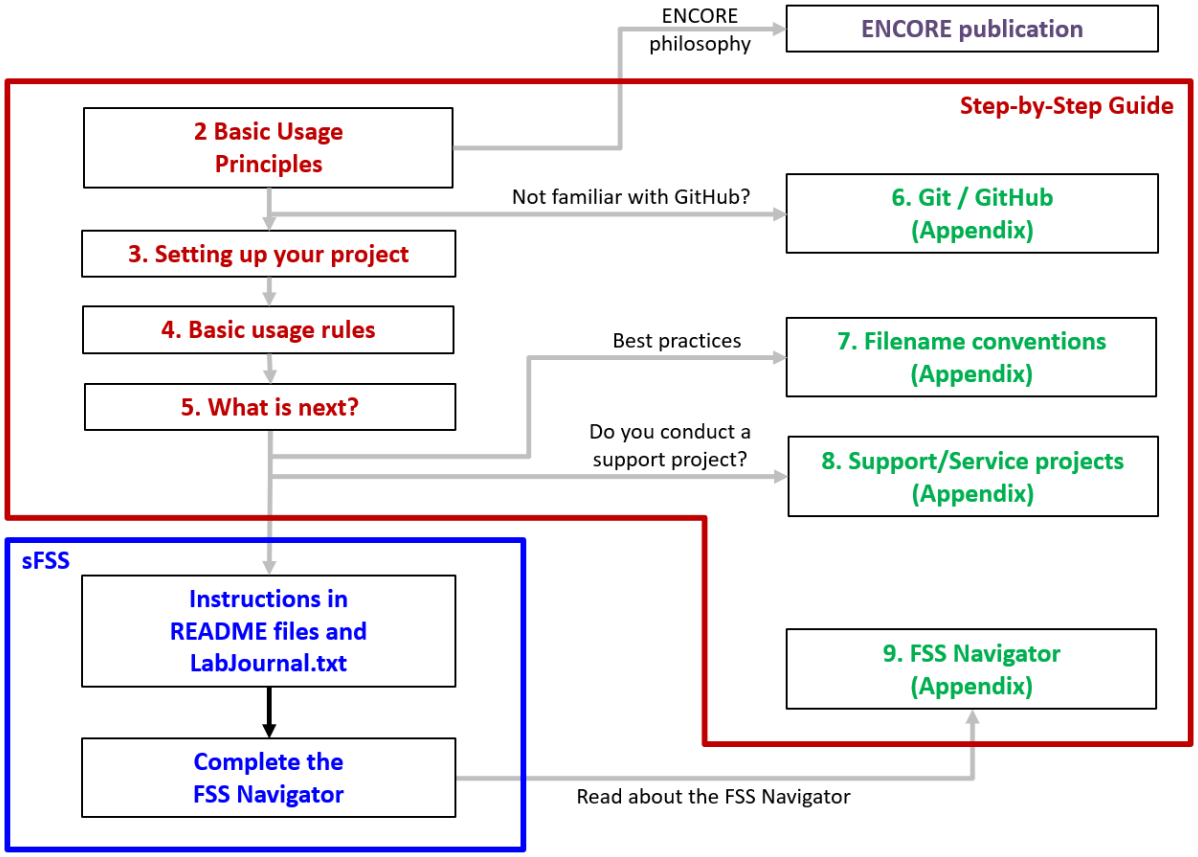

Figure 2. This flow diagram guides you through the documentation. The sections inside the brown boxes are part of this Step-By-Step guide. The sections indicated in red should be read and followed. The sections indicated in green are optional. Once you have set up your project, you will find more specific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

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. *In prep.*

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 (2023) ENCORE. A case study in spatial transcriptomics. *In prep.*

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 FSS Navigator. There are alternative ways of doing this, but this will get you going.

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 sFSS 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 FSS 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 FSS NAVIGATOR

The FSS 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. *Figure 2* shows an example of an **Navigate.html** reflecting the FSS Navigator project itself. To setup the FSS navigator, proceed as follows:

1. The FSS 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 FSS 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**\00_README-FIRST.{md.txt}** for instructions.

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

Note
The FSS 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.

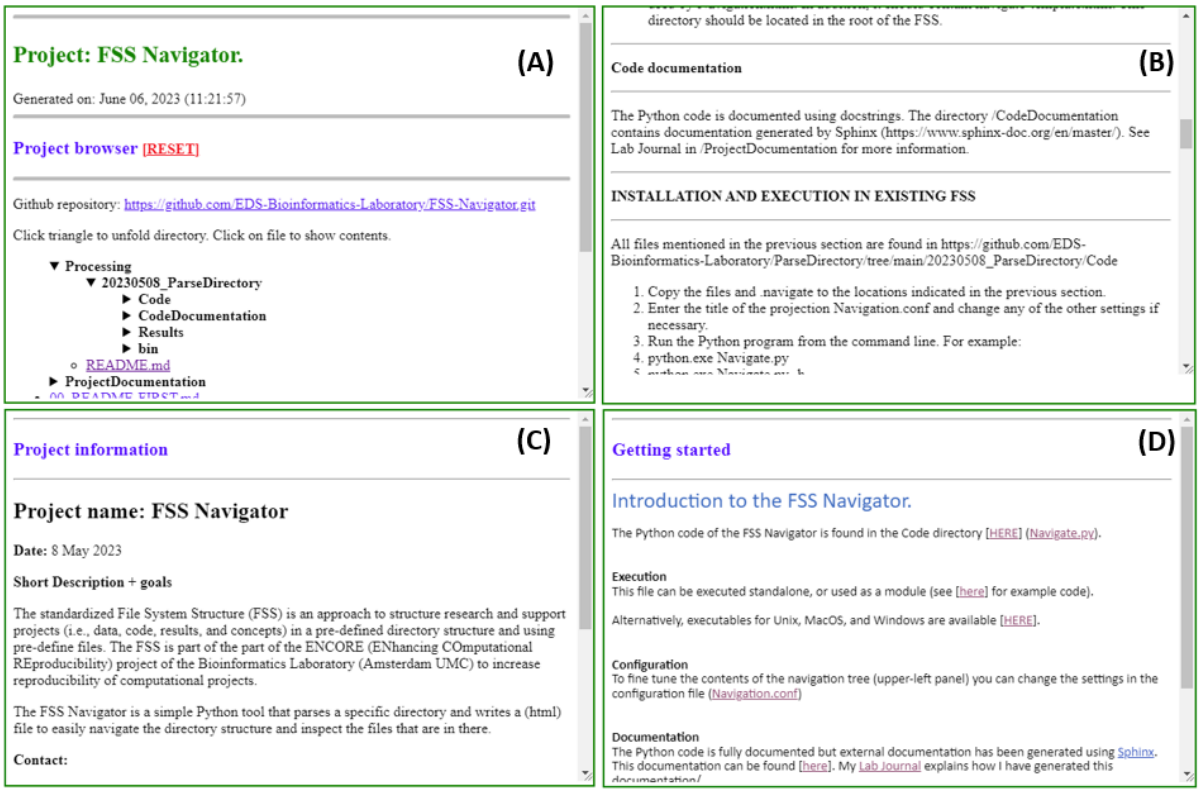

Figure 2. **FSS Navigator for the FSS Navigator project**. Web-browser showing Navigate.html for the FSS Navigator project. (A) Expandable sFSS directory tree and link to the GitHub repository. The project owner can configure which directories and files to show. (B) Content of selected file. In this 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 directly 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 sFSS 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 FSS navigator binaries. Changes between versions are documented in GitHub Issues and with the specific releases.

- **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
    - Directories and files that are not used can be removed but don't remove the following files in the sFSS root directory:
        - 00_README-FIRST.md
        - 1_Step-by-Step-ENCORE-Guide.pdf
        - 2_CITATION.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 FSS 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. Browse through the various sFSS directories and consult the 0_README.md, the LabJournal.txt, and other files for specific instructions about the information you need to provide in each sub-directory.

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

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

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 (*Figure A.1*)

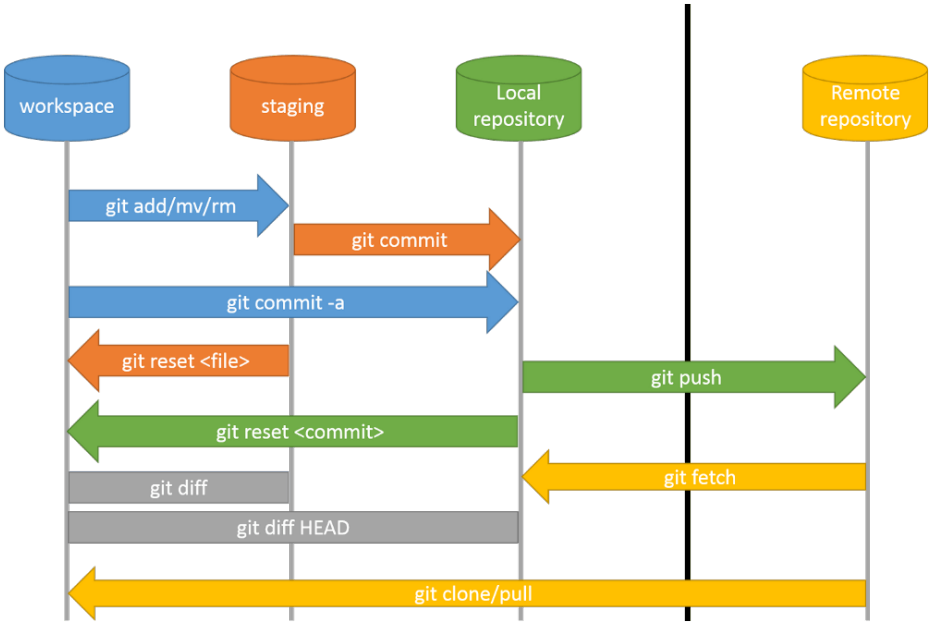

Figure A.1. The overall architecture of the git/GitHub environment.

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 (*Figure A.2*).

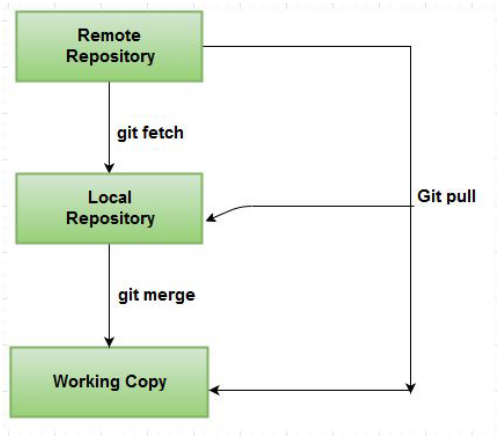

Figure A.2. Git fetch vs Git pull.

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 FSS 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 FSS Navigator was developed to provide a first guidance through the project. The FSS Navigator itself was developed following the ENCORE approach, and the project package is found in Zenodo (https://doi.org/10.5281/zenodo.7985655). The FSS 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 (Figure 2). 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 FSS Navigator is work in progress and updates will become available in the future.

The following files are part of the FSS navigator:

- **Navigate.html**. Open in your web browser to navigate the sFSS.
- **Navigate.py**. Standalone Python 3 script to generate Navigate.html to navigate the FSS. 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
- There are also executables available for Windows and MacOS and are found at Zenodo:
    - DOI: https://doi.org/10.5281/zenodo.7985655; https://zenodo.org/record/7985655))
    - **Navigate_W.exe**. Windows executable if you don'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**. Example Python script to show how to include Navigate.py as a module in other Python scripts. This may help to automatically keep Navigate.html up-to-date without manually executing Navigate.py.
- **Navigate.conf**. Configuration file for Navigate.
    - The title of your project should be specified in this file.

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

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.

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