ENCORE: ENhancing COmputational REproducibility

A Bioinformatics Laboratory initiative

Date: 28 June 2023 **Version:** 11b	**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 Research initiative of the Bioinformatics Laboratory. Each project that follows the ENCORE principles comprises three main components: the standardized File System Structure 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 found in the README.md (and other files) that you will find in the sFSS.

The standardized File System Structure (sFSS): This provides a standardized directory structure (template) according to which your project should be organized. It provides a section for the background information, data, and processing. Roughly, it has the structure shown in *Figure 1*, which is explained in more detail below and in the documentation contained within the sFSS.

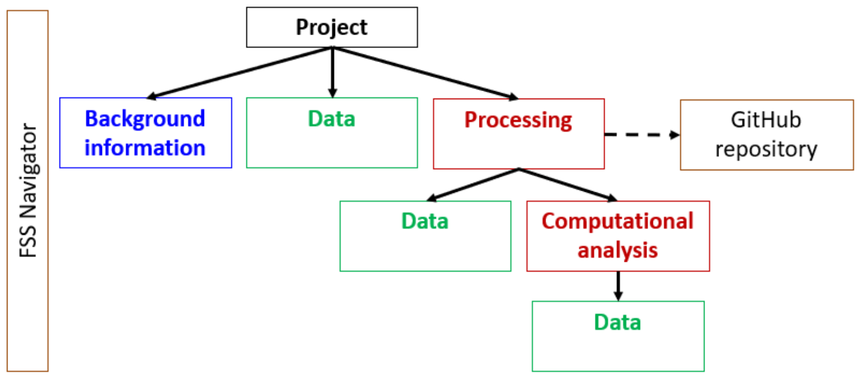


Figure 1. The overall structure of the standardized File System Structure (sFSS).

The GitHub Repository: Git and GitHub provide a system for software version control, hosting, and sharing (Appendix 1)**.**

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.

2 REFERENCES

- Van Kampen AHC, Mahamune U.M, …….., Jongejan, A, Moerland, PD (2023) ENCORE: Enhancing Computational Reproducibility, in prep.
- Van Kampen AHC, Mahamune U, Jongejan, A (2023) The standardized file system structure (FSS) navigator. Zenodo. DOI: https://doi.org/10.5281/zenodo.7985655
- Mahamune U, Moerland, PD, Van Kampen AHC (2023) ENCORE. A case study in spatial transcriptomics. In prep.

3 ORGANIZING YOUR PROJECT: THE RECIPE

If you are new to GitHub then first read *Appendix 1* and make sure you have a **GitHub account** and to **install git bash**. The recipe below will take you step by step towards the creation of a new GitHub repository and sFSS for your project. There are alternative ways of doing this, but this will get you going.

3.1 CREATE YOUR PROJECT REPOSITORY (REPO) DIRECTLY ON THE GITHUB WEBSITE

In this first step you will create a GitHub repository on github.com.

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., **ComputationalProject**).
   - 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 keywords.
    - Standard keywords are: research, support, education.
    - Look at other repositories for used keywords to keep consistency
    - *Rationale:* having keywords helps to retrieve specific repositories.

Now you are ready to use this repository from your own computer/laptop using e.g., git bash (see *Appendix 1*).

3.2 COPY THE GITHUB REPOSITORY 'REPRODUCIBILITY' IN THIS DIRECTORY

The next step is to copy the sFSS template to your own computer.

1. Start git bash in the directory where you want to create your project.
2. Download the sFSS template from the ENCORE GitHub Repository using the next command in git bash:
    - git clone https://github.com/EDS-Bioinformatics-Laboratory/ENCORE
3. This will create the directory ENCORE, which contains the sFSS.
4. Rename ENCORE to the name of your project (e.g., **20231201_Project**)
    - 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).

Now you can start using the sFSS and GitHub/Git for your project.

Note about the Markdown files

In the sFSS you will find markdown files (file extension .md). If you are not familiar with this 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).

3.3 INITIALIZE YOUR SFSS PROJECT

Next you should start populating your sFSS project (**20231201_Project**) by doing some administrative work.

1. Enter information in the **0_PROJECT.md** file in the FSS root directory
    - See instructions in this file.
    - This file is used by the FSS Navigator (see below).
2. Enter information in the 20231201_Project**\Processing\README.md**.
    - 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 a next steps).
3. Edit github.txt
    - Add the name of your GitHub repository (e.g., https://github.com/YourAccount/ComputationalProject.git**)** to 20231201_PROJECT**\Processing\github.txt**
    - This file is used by the FSS Navigator (see below).
4. 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.
    - See instructions in 20231201_Project**\Processing\README.md.**
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, and update whenever needed.
6. Start your lab journal
    - See 20231201_Project**\ProjectDocumentation\labjournal.txt** for instructions.

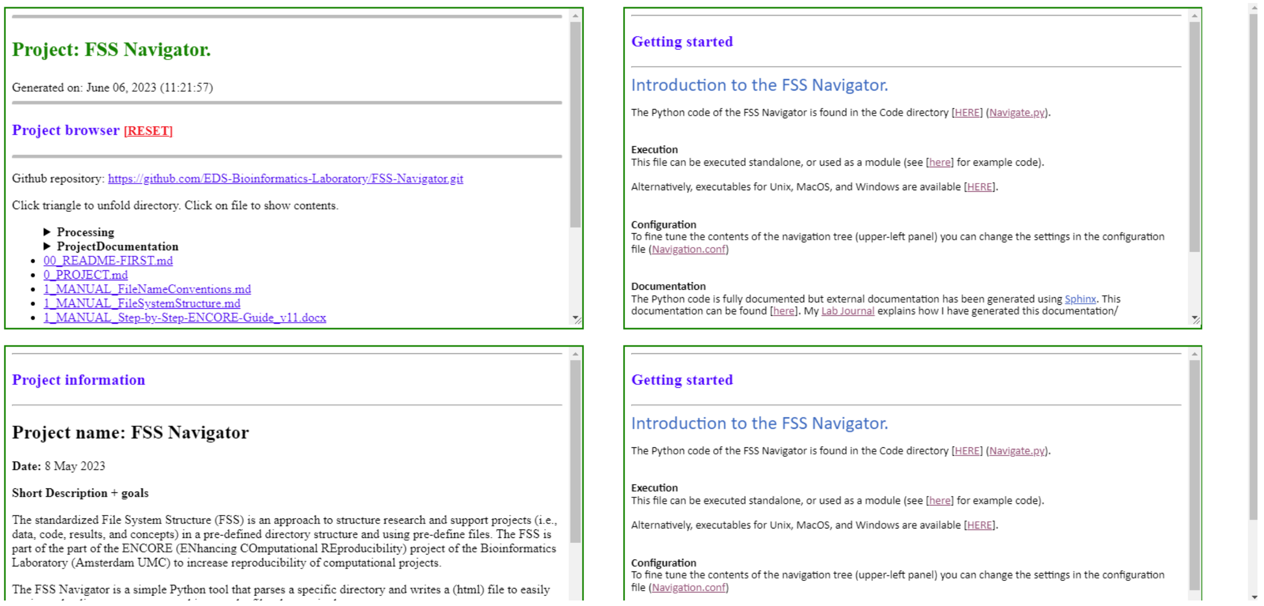
3.4 SETUP THE FSS NAVIGATOR

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. *Figure 2* shows the sFSS of the FSS Navigator itself. The FSS Navigator project can be downloaded from https://doi.org/10.5281/zenodo.7985655. Note, that the FSS Navigator is work in progress and updates will become available in the future.

The FSS Navigator can be configured using 20231201_Project**\Navigation.conf**.

The FSS Navigator uses 20231201_Project**0_GETTINGSTARTED.html**. This is the default file which contains instructions once you want to modify this file (typically near the end of the project).

To generate the 20231201_Project**\Navigate.html** file, you need to run 20231201_Project**\Navigate.py**. See 20231201_Project**\00_README-FIRST.{md.txt}** for instructions.



Project: FSS Navigator.

Generated on: June 06, 2023 (11:21:57)

Project browser [RESET]

Github repository: https://github.com/EDS-Bioinformatics-Laboratory/FSS-Navigator.git

Click triangle to unfold directory. Click on file to show contents.

▶ **Processing**
▶ **ProjectDocumentation**

- 00_README-FIRST.md
- 0_PROJECT.md
- 1_MANUAL_FileNameConventions.md
- 1_MANUAL_FileSystemStructure.md
- 1_MANUAL_Step-by-Step-ENCORE-Guide_v11.docx

Getting started

Introduction to the FSS Navigator.

The Python code of the FSS Navigator is found in the Code directory [HERE] (Navigate.py).

Execution
This file can be executed standalone, or used as a module (see [here] for example code).

Alternatively, executables for Unix, MacOS, and Windows are available [HERE].

Configuration
To fine tune the contents of the navigation tree (upper-left panel) you can change the settings in the configuration file (Navigation.conf)

Documentation
The Python code is fully documented but external documentation has been generated using Sphinx. This documentation can be found [here]. My Lab Journal explains how I have generated this documentation/

Project information

Project name: FSS Navigator

Date: 8 May 2023

Short Description + goals

The standardized File System Structure (FSS) is an approach to structure research and support projects (i.e., data, code, results, and concepts) in a pre-defined directory structure and using pre-define files. The FSS is part of the part of the ENCORE (ENhancing COmputational REproducibility) project of the Bioinformatics Laboratory (Amsterdam UMC) to increase reproducibility of computational projects.

The FSS Navigator is a simple Python tool that parses a specific directory and writes a (html) file to easily

Getting started

Introduction to the FSS Navigator.

The Python code of the FSS Navigator is found in the Code directory [HERE] (Navigate.py).

Execution
This file can be executed standalone, or used as a module (see [here] for example code).

Alternatively, executables for Unix, MacOS, and Windows are available [HERE].

Configuration
To fine tune the contents of the navigation tree (upper-left panel) you can change the settings in the configuration file (Navigation.conf)

Figure 2. Web-browser showing Navigate.html for the FSS Navigator project. The top-left panel shows an expandable hierarchy of the sFSS directories. The lower-left panel shows 0_PROJECT.md. The lower-bottom panel shows 0_GETTINGSTARTED.html. The upper right panel is used to show the content of files within the sFSS.

3.5 SYNCHRONIZE YOUR FSS 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/ComputationalProject.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 (in Microsoft Windows: right mouse click, then select 'Git bash here')
3. Enter the following git commands (after each command you can use git status to check):
    - git init –initial-branch=main
    - git remote add origin [*URL of repo*]
        - URL of repo: as entered in github.txt, e.g.,
        - https://github.com/YourAccount/ComputationalProject.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/ComputationalProject.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.

3.6 START USING YOUR SFSS

Congratulations! You have now setup 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 about the best practices using the sFSS and its pre-defined files (Appendix 2).
2. Read the ENCORE publication.
3. Browse through the various sFSS directories and consult the 0_README.md files for further 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 ON A DAILY BASIS!!**

3.7 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 daily/weekly basis).

1. Go to 20231201_PROJECT**\Processing**.
2. git pull https://github.com/YourAccount/ComputationalProject.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

4 USING GITHUB AND GIT

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*)

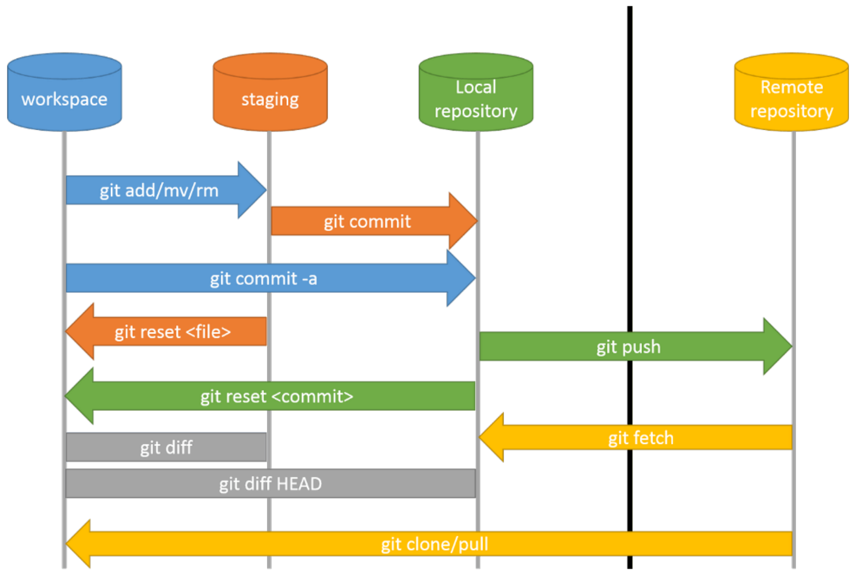


Figure A.1. The overall architecture of the git/GitHub environment.

4.1 GITHUB ACCOUNT

If you do net yet have a GitHub account then visit their website a https://github.com, and sign up for GitHub.

4.2 GITHUB

GitHub is used to manage your software. You have been invited to the EDS GitHub Repository by your supervisor. This is an **organization repository** on GitHub.com. Once you accept the invitation you will have access to all software currently being developed by the Bioinformatics Laboratory. Most of the software repositories are private (not public) and you are not allowed to redistribute any of the repositories since they may contain confidential information.

Location: https://github.com/EDS-Bioinformatics-Laboratory

4.3 INSTALL GIT BASH

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.

4.4 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

[add refs] (Blischak et al., 2016; Deardorff, 2020; Perez-Riverol et al., 2016; Ram, 2013).

4.5 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
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 9e then it is likely that something went wrong with the authentication.
11. Go back to GibHub in your webbrowser 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

4.6 FURTHER GIT/GITHUB NOTES

It is not the intention of this Guide to give a full overview of all git/GitHub scenarios and commands. However, you mind find the information below useful in case you run into problems synchronizing your sFSS with GitHub.

4.6.1 GIT PULL VS GIT FETCH

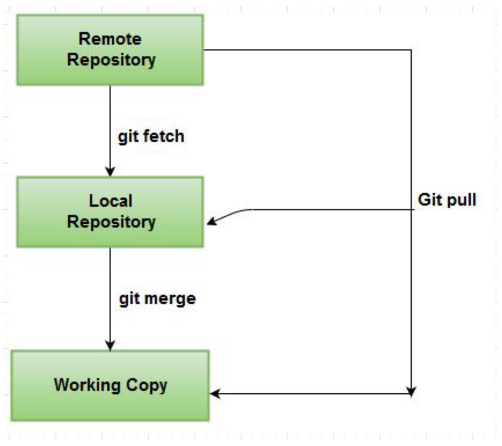


Figure A.2. Git fetch vs Git pull.

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*).

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

4.6.2 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.

4.6.2.1 OVERVIEW OF THE BRANCHES ON YOUR WORKSTATION

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

4.6.2.2 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

4.6.2.3 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

4.6.2.4 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

4.6.3 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).

4.6.4 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

4.6.5 PROBLEMS WITH 'MERGING'

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

```
git mergetools
```

4.6.6 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
```

4.6.7 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.

4.6.8 AUTOMATIC CREATION OF REPOSITORY FROM WITHIN R/RSTUDIO

Aldo Jongejan has a R script that creates a new repository for you when working from R/RStudio. You wouldn't have to go to the GitHub website first to create a new repository.

https://github.com/aldojongejan/R_tests.git - zie getFSS.r

Example in R:

```
# Start new analysis
source("getFSS.r")
startNewRepo("20210119_FleurPeters_CLL_Dasatinib", analysisDir = "Dasatinib")
```

5 APPENDIX. BEST PRACTICES FOR USING THE STANDARDIZED FSS AND PRE-DEFINED FILES

5.1 TEMPLATE

- **Location.** For new projects the latest ENCORE template should be downloaded from GitHub (https://github.com/EDS-Bioinformatics-Laboratory/ENCORE).
- **Updates.** ENCORE is still evolving and updates in the FSS structure and pre-defined files can be expected with increasing experience and extensions.
- **Automatic instantiation.** In case you developed software to automatically instantiate a new ENCORE project then the software should retrieve the latest ENCORE template, or generate the FSS structure including the pre-defined files (including their content) automatically.
- **Directory and file names.** The FSS subdirectory names and structure, and the pre-defined file names should not be changed to ensure consistency across projects.
    - Don't add/remove dashes, spaces.
    - You are allowed to add/remove a '0_' prefix to/from the name names to ensure that file is at the top of the file list.
    - Additional subdirectories may be added to the FSS if needed to get a better organization of the project but the overall base structure and use should remain as is.
    - Directories and files that are not used can be removed.

5.2 GENERAL

- **Project name.** Project name should include year, month, day (YYYYMMDD_ProjectName) or project identifier (ID_ProjectName).
- **File names.** Name all project files to reflect their content or function.
- **Person names.** If you use names of persons in any part of your documentation then ensure that the first and last name, titles, and affiliation are documented once (since the FSS may be shared with others). The project leader and project collaborators should be added to the 0_PROJECT.md file of the top-level directory such that additional information (affiliation) and roles of these persons can be retrieved.
- **FSS Hub.** The FSS is the entry point of a project, not GitHub. Consequently, all project information should be stored within the FSS. Basically, we use GitHub only internally for versioning. The FSS contains the software version that is shared. However, in general, we will also provide access to the GitHub repository when sharing the FSS.
- **Backup.** Make (incremental) backups of your project on a daily basis (e.g., to an external harddisk and/or Cloud).

5.3 MAINTAINING YOUR FSS

- **Conceptual information** should provide information the concepts and approaches used in the project to increase transparency and enable the full understanding of the computational project. Consequently, this information should provide important part of the documentation that is found in the README files that describe data, software, and results. The documentation in the README files can be complemented by oral/poster project presentations, scientific literature, the lab journal, and other relevant (background) information.
    - For example, the documentation should describe the applied computational methods, an explanation of why the computational experiments were done, how data was derived in the wet-lab, how and why the data was pre-processed, observations that you make about the data and results, relevant thought processes, etc. References to relevant resources (e.g., data, original source of data) should be supplied as much as possible.
- **Updating**. Keep your FSS and pre-defined files up to date on a daily basis during the project. Don't attempt to do this at the end of the project.
    - Keeping up-to-date during the project allows the supervisor and collaborators in a project to always inspect, understand, and reproduce the content of the FSS at any point in time.
    - It is impossible to recall all project details from memory and document these once the project is finished, published, or archived.
    - The FFS and pre-defined files (e.g., labjournal.docx, 0_PROJECT.md, and 0_REAMDE.md) should enable your peers to understand and reproduce your project. Document your project with this in mind.
    - Document everything in detail such that you can re-use for publication or presentation. The FSS will, generally, contain more detailed information than found in a publication (e.g., space restrictions).
    - When providing background information, and describing data, code, and results make explicit cross-references as much as possible. For example, link code to data in the description.
- **README files.** Most directories contain a 0_README.md file. This file should be used to clarify the content of the directory in detail.
    - All README files contain instructions in *italics*. These instructions can be removed once you completed the README file.
    - The README file contains a basic template to guide you in providing the required information. However, if necessary, more information should be provided.
    - If you prefer to another file format (e.g., Word, LaTex) instead of markdown, then change the 0_README.md file accordingly.
    - 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.
- **Self-contained.** Each FSS project should be self-contained and references to any item (including references made to files from the software) should be relative to the FSS root to ensure all code remains functional if copied to a different location.
- **Don't maintain documentation outside the FSS**. All project documentation should be kept inside the FSS. Consequently, don't keep separate documentation or relevant information/discussions in email archives, paper notes, Slack, Whatsapp, etc. This should all be copied by the project owner to the LabJournal or into other pre-defined files.

- **Location.** 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) to contains all documentation.

6 ENCORE FOR SUPPORT PROJECTS

- For support projects it is not required to sync the Processing directory for every individual project to GitHub

6.1 USING BRANCHES FOR SUPPORT

I tested the branch function for a support project. Normally I run stable code and, in that case, I only need to store the version of the stable code and the parameters that were used. However, I also use Python notebooks where I make changes in the code. E.g., a file can be comma-delimited instead of tab-delimited. Column names might be different, so then I need to change the code as well.

I created a new branch, from the master branch, with the name "runXXX-20201005-maria-reseda20201006". After I did all the analysis, I committed all the changes and pushed it to GitHub. I will probably never use this branch again, but now I have a record of the changes that I made. For that reason, I will not merge this branch into the master branch! I also included a "README-analysis.md" file in this branch where I tried to record as much as possible what I did.

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. 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 June 2023

- Re-written and re-structured this Guide, and improved layout.
- Added an introduction
- Added a section about sFSS best practices

9 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