welcome booklet

PROVitD team

2026-01-30

Table of contents

1 Introduction

This book contains information needed by a new comer in the PROVitD Group regarding data management in the lab, good practice in R, ressources on bioinformatics topics…

1.1 Contributing

May you have any suggestion, find any typo, or anything want to make a change that could improve this welcome booklet, let us know by creating an issue or by editing the page and creating a merge request. It is available on the right side of any page in this quarto book.

Please keep in mind that this is a public booklet, so do not add any confidential information.

Part I

Good practice

2 Raw data

2.1 What is raw data?

In bioinformatics, raw data are unprocessed output from a specific technology. It is data that you cannot regenerate (or not easily), and that should never be modified. The type of files can vary, but a good exemple for sequencing technologies is `.fastq` files.

Sometimes, the sequencing platform will also give you some pre- or fully analyzed data. An example in the case of single-cell RNA-seq is cell-by-gene matrices. If it is the basis of your analysis, you should consider them just like raw data.

These files are required to publish.

2.2 Where to save raw data?

At the IGBMC, there are two storage infrastructures for scientific data, their usage is described here, but as a reminder:

Storage	Usage	Replication	Daily snapshots
`mendel`	Data analysis, active projects	Yes	Kept 30 days
`space2`	Data conservation, finished projects	No	Kept for 3 days

So in practice, you should save your raw and analyzed data to a `mendel` space.

Once your analysis is finished and ready for publication, you should upload them on a database, and you can now leave them on `space2`.

Once your paper is published, and the data is accessible online for all (e.g. on GEO), you can remove from `space2` any file that is now accessible online. But do add a `.txt` file that specifies how the data can be retrieved (e.g an accession code).

2.3 Steps to save raw data

So here are the general steps on how to correctly save raw data:

- Download the data and store them securely: Save them on a mendel space or a backed-up institutional storage. Do not save your data only on your local computer, if it is broken, lost, or stolen, the data will be lost forever.
- Verify its integrity: Along with your data, you should have a way to verify the data was not corrupted during its download. For example, you might be given the md5 of files. You must always keep the file containing the original md5 values together with the data.

Note

A md5 is 32-character fingerprint of the file. If the file is not exactly the same as the original one, recomputing the md5 will give a different result and you will know that your file was corrupted (i.e. modified, truncated…).

Tip

In R, you can for example run `tools::md5sum("/path/to/file")` to get the md5 of a file.

- Do not touch it anymore: Make the files only readable. You can go to the Properties of a file and set its permission to `Read only`. Or using command line, you can restrain the modification of a folder with `chmod -r 444 /path/to/folder`. This also includes not changing the name of the files! If you need to put some data in another folder, or with another name, you can create symbolic links with `ln -s` in bash, or a shortcut file.

2.4 What about metadata?

Data is a collection of raw and unorganized information. Without proper processing, organizing, and information about it, it cannot be interpreted. Metadata is data about data. It gives a description and a context about the data. It can be the species, tissue type or conditions of the samples, as well as the lab protocol, sequencing technology etc…

If you filled GenomEast's LIMS correctly, you already have some of that information. You can save the project page of the study as a pdf and put it in the folder that contains the data. It contains general information about the study (ID, title, technology, date, collaborators, aim, description of samples…). You can also fill and export the table of the Samples conditions in `.csv` format.

A metadata file for your samples can be in a excel, `.csv`, `.tsv`, `.txt` format, or any format that can be easily exploited in further analysis in a programming language. What is important that:

- each samples have a unique identifier
- one variable corresponds to one column
- have consistant values inside of the column

For example:

```
,Condition_treatment
Sample1,Control_6
Sample2,TreatmentA_6
Sample3,TreatmentB_6
Sample4,ctrl_9
Sample5,TreatmentA_6
Sample6,TreatmentB_6
```

is bad! There are two variables in the same column (`Condition_treatment` should be seperated into `Condition` and `Treatment`), and the values in this column are not consistent (`Control` is not the same as `cntrl`).

```
,Condition,Age
Sample1,Control,6
Sample2,TreatmentA,6
Sample3,TreatmentB,6
Sample4,Control,9
Sample5,TreatmentA,9
Sample6,TreatmentB,9
```

is good.

3 Organize an analysis

3.1 Generalities

In the lab, we organize analysis based on projects (`2025_scRNA_patient_prostate`), not based on people (`JaneDoe_scRNA`). It helps unsure that even after a person leaves the lab, we can retrieve their analysis easily.

We aim to keep the same organization inside folders. This organisation is still a work in progress. If you have suggestions to make it better, please share with the lab your ideas!

An analysis should be breakable in a couple of folders:

- `data/` it should contain (links) to raw data. It also contains useful metadata. A link can be created in R like so: `R.utils::createLink(link, target)`
- `script/` it should contain one or several (organised) scripts that can be run from start to finish to reproduce your analysis. Create folders for specific subanalysis with the date as a prefix (e.g `YYMMDD_trajectory_Luminal`) make sure to use the same structure in the `result/` folder. A folder can be created in R like so: `dir.create(directory)`
- `result/` it should contain tables, plots, R object that were produced during the analysis. You can create specific folder to keep a nice structure, for example `figure`, `table`, `rds`…
- `doc/` containing various documentation that needs to be save, for example important related papers

An example of an analysis folder could be:

```
data
   ABCetall_2021_scRNA -> ../../../litterature/ABCetal_2021_scRNA/
   S19192_scRNA -> ../../../projects/S19192/
   S22024_S22194_snATACseq -> ../../../projects/S22024_S22194_snATACseq/
   S23036_spatial -> ../../../projects/S23036_CytAssist
doc
   presentation
       200314-scRNA-labmeeting.pdf
       230122-snATAC-labmeeting.pdf
       230701-snATAC-congress.pdf
```

```
        231214-spatial-labmeeting.pdf
    litterature
        ABCetal_2021.pdf
        XYZetal_2019.pdf
analysis
    200301_scRNA
        200301_preprocessing
            preprocessing.qmd
            preprocessing.html
            rds
                original_seurat.rds
                preprocessed_seurat.rds
            figure
                PCAPlot.png
                UMAPPlot.png
                VlnPlot_postfilter.png
                VlnPlot_prefilter.png
        200304_integration
            integration.qmd
            integration.html
            rds
                integrated_seurat.rds
            figure
                UMAPPlot_integrated.png
        200306_cell_annotation
            cell_annotation.qmd
            cell_annotation.html
            rds
                annotated_seurat.rds
            figure
                UMAPPlot_CellSubType.csv
                UMAPPlot_CellType.png
            table
                cell_annotation_CellSubType.csv
                cell_annotation_CellType.csv
                FindAllMarkers_CellSubType.csv
                FindAllMarkers_CellType.csv
        200308_trajectory_Luminal
            trajectory_Luminal.qmd
            trajectory_Luminal.html
            rds
                trajectory_Luminal.rds
            figure
```

```
                trajectory_Luminal.png
    230103_snATAC
     ...
    231202_spatial
     ...
README.txt
```

3.2 Data

Do not copy all of the data to your `data/` folder if it is already stored somewhere. It saves space (as there won't be several copy of the same heavy raw data), and makes sure that there is only one place where the raw data can be found to avoid confusion. Remember, you should never modify raw data files. Instead, use a shortchut, or a link, to the folder. This can be done in bash like so:

```
ln -s <target> <symlink>
```

where you remplace by the folder from which to create the link, and by the name to give the symbolic link (or shortcut), e.g. `ln -s S19192_scRNA ../../../projects/S19192/`

or in R like so:

```
createLink(link=symlink, target)
```

where you remplace `target` by the folder from which to create the link, and `symlink` by the name to give the symbolic link (or shortcut), e.g. `createLink(link="S19192_scRNA", "../../../projects/S19192/")`

Useful metadata should already be associated with the raw data.

3.3 Analysis

In the `analysis/` folder, the general rule is to have one folder per subanalysis. The subanalysis folder is named with the date and a general title (e.g. `200301_scRNA/`). You can create as many subanalysis and subfolders as necessary (e.g. `200301_scRNA/200301_preprocessing/`). In each subanalysis folder, there is one script, most likely a quarto document `.qmd` that creates an easily readable report in `.html`. Any result created by this script is present in the same folder. If many results are created, organise them into folders (e.g. `rds/` for saving R objects, `figure/` for saving plots, `table/` for saving tables etc.).

Think about the architecture of your folder before starting your analysis. For this you need to break down your general idea of the analysis into smaller chunks. Are you going to analysis one or several datasets (and so, create one folder per dataset)? What are the planned steps of your analysis (and so, create one folder per step for a given dataset)? Of course, you might not know all of the analyses you will do in advance, but you will get an idea of the level of complexity of the architecture of folders to create.

This is important as it is not a good practice to move and rename folders or files. Some scripts depend on the results of previous analyses, so moving and renaming folders or files can create discrepencies in other analyses.

Part II

R tips

4 R projects

Working with R Projects is a good practice as it provides a structured and efficient way to manage your R scripts, data, and other files. Here are some reasons why you should always be using R Projects: Working with R Projects is a good practice when analysing in R. It provides a structured and efficient way to manage your R scripts, data, and results files, ensuring that your analysis is reproducible, organized, and easy to share.

Here are the key reasons why you should always use R Projects:

- Better organization: All related files (scripts, raw data, results, documentation) are stored in one single folder. This prevents confusion and makes it easy to navigate your work.
- Automatic setting of working directory: R Projects automatically set the working directory to your project folder every time you open it. This means your file paths will work consistently, avoiding frustrating "file not found" errors.
- Easier collaboration: When sharing a project folder, collaborators get all the files and can run your analysis without reconfiguring file paths.
- Compatibility with best practices as R Projects integrate smoothly with:
  - Git for version control, which tracks changes, allows you to roll back to earlier versions, and collaborate effectively.
  - renv for package management, which records the exact package versions used so your project can be reproduced exactly in the future or on another computer.

5 Renv

5.1 What is renv?

`renv` is a package management system for R that helps ensure reproducibility and portability of R projects. It allows you to create isolated environments for your R projects, making it easier to share and collaborate on code. The documentation is available at: https://rstudio.github.io/renv/articles/renv.html.

5.2 Why should you use renv?

For different analysis you might rely on various R packages. However, managing package versions and dependencies can be challenging. Here's why `renv` is a good practice for biologists:

1. Reproducibility: `renv` allows you to capture the exact versions of R packages used in your project. This ensures that your code can be reproduced in the future, even if new package versions become available.
2. Collaboration: With `renv`, you can more easily share your R projects with colleagues or collaborators. They can set up the same environment using the `renv.lock` file, ensuring that everyone is working with the same package versions.
3. Avoid conflicts: By using isolated environments `renv` makes it easier to avoid conflicts between different package versions, ensuring that your code runs consistently, even if you need to update a package for another analysis.

6 Implementing renv in your R projects

6.1 Installing renv

To start using `renv`, you need to install it first. Open your R console and run the following command:

```
install.packages("renv")
```

6.2 Working with R projects

Working with R Projects in RStudio is necessary to use `renv`. But it is a good practice in general as it provides a structured and efficient way to manage your R scripts, data, and other files. Here are some reasons why you should always be using R Projects:

1. Organization: R Projects help you keep all related files in one place. This makes it easier to manage your scripts, data, and outputs, ensuring that your work is well-organized.
2. Reproducibility: R Projects ensures that your working directory is set correctly every time you open the project. This helps avoid issues related to file paths and makes your analysis reproducible.
3. Collaboration: R Projects make it easier to collaborate with others. By sharing the project folder, your collaborators can easily access all the necessary files and run the analysis without any issues.
4. Version Control: R Projects integrate seamlessly with version control systems like Git. This allows you to track changes, revert to previous versions, and collaborate more effectively with others.
5. Environment Management: R Projects allow you to manage your R environment more effectively. You can use packages like `renv` to create isolated environments for each project, ensuring that package versions and dependencies are consistent.

How to Create an R Project, you ask?

Creating an R Project in RStudio is simple:

1. Open RStudio.
2. Go to `File` > `New Project`.
3. Choose `New Directory` or `Existing Directory` depending on your needs.
4. Follow the prompts to set up your project.

6.3 Initializing renv in a project

Once `renv` is installed, you can initialize it in your R project. Open your R console, navigate to your project directory, and run the following command:

```
renv::init()
```

This will modify the `.Rprofile` file, as well as create an `renv/` directory and a `renv.lock` file in your project, which will be used to manage your project's environment as follows:

- `renv.lock`: This file captures the exact versions of packages used in your project. It ensures that the same package versions are used when the project is shared or restored.
- `renv/` folder: This folder contains the isolated environment for your project. It stores the packages and their dependencies specific to your project.

6.4 Managing packages with renv

To add packages to your project, you can use the same usual functions, e.g.:

```
install.packages("dplyr")
BiocManager::install("DESeq2")
```

This will install the package in your project's environment only. Once you have installed the packages that you want, you can update the `renv.lock` file by running:

```
renv::snapshot()
```

You can run this command at any time, for example when you add or update some packages.

At some point you might use one of the two following commands:

- the `renv::status()` command compares the current state of the project's environment with the `renv.lock` file. It identifies any changes made to the environment, such as installing or updating packages. This helps you track the modifications made to the environment and ensures that the project remains reproducible.

- the `renv::restore()` command restores the project's environment to the state captured in the `renv.lock` file. It installs the exact package versions specified in the file, ensuring that the project can be reproduced with the same environment. This is useful when you want to recreate the environment on a different system or restore it after modifications.

6.5 Sharing your project with renv

To share your project with others, you can share the `renv.lock` file. Remember to run the `renv::snapshot()` command to capture the current versions of packages used in your project.

6.6 Recreating a renv environment from a `renv.lock` file

To recreate the same environment using the `renv.lock` file, you can use the `renv::restore()` function. Here are the steps to do this:

1. Navigate to a new or desired R project.
2. Make sure the `renv.lock` file is located in the R project directory.
3. Execute the following command to restore the environment:

```r
renv::restore()
```

This command will read the `renv.lock` file and install the exact versions of the packages specified in the file. It will recreate the environment as it was when the `renv.lock` file was created.

7 Conclusion

Using `renv` in your R projects as a biologist can greatly improve reproducibility, collaboration, and portability. By managing package versions and dependencies, you can ensure that your code remains consistent and can be easily shared and reproduced. Find out more about `renv` usage by reading their documentation at https://rstudio.github.io/renv/articles/renv.html.

Part III

Bioinformatics ressources

8 Single Cell

8.1 Packages

- R:
  - General analysis:
    - Seurat
  - Annotation:
    - sc-type
    - SingleR
  - Trajectories:
    - dynverse
    - monocle3
    - slingshot
  - Spatial analysis:
    - semla
- Python:
  - General analysis:
    - scanpy
    - anndata

8.2 Documentation

- Best practices in single cell [link]
- Tools developped for single cell [link]

8.3 Databases

- cellxgene
- Human Cell Atlas

- EBI Single Cell Expression Atlas
- Arc Virtual Cell Atlas: 230 million cells (and expanding), spanning 21 organisms and 72 tissues

9 Workshop and training

9.1 In-person

- Non-exhaustive catalog of trainings in France [link]
- Well recognized 1-week training to NGS and coding in Roscoff [link]
- Catalog of trainings offered by the Swiss Institute of Bioinformatics [link]

9.2 Online

- Teaching material from GenomEast trainings [link]
- The Carpentries teached workshops on R/RStudio/bash applied to genomics [link]
- Teaching material from Harvard Chan Bioinformatics Core [link]
- A large catalog of educational material for bioinformatics [link]

10 Cheat Sheets

You can also follow the links to various cheat sheets and print them out to decorate your desk:

- R: to discover R universe
  - Base R: to get started with R
  - RStudio: to get started with RStudio
  - R Best Practice: to improve your R skills
  - ggplot2: to get started with generating plots
  - dplyr: to manipulate datatables
  - tidyr: to tidy datatables
  - quarto: to create beautiful notebooks with integrated code
  - rmarkdown: to create beautiful notebooks with integrated code (deprecated)
- git: to version control code
  - git branching model: to develop tools with version control
- conda: to use contained and reproducible package environment