Version Control III

Joachim Vandekerckhove

Winter 2025

File structure and naming conventions

A successful file structure organizes your data and code with the goal of repeatability, making it easier for you and your collaborators to revisit, revise and develop your project.

File structures help communicate the function and purpose of elements within a project by separating concerns into a hierarchy of folders and using consistent, chronological, and descriptive names.

Begin with something simple and clear to avoid bogging down at the start, and allow the structure to evolve as needed.

Why file organization matters

Consistency matters!

Like navigating around the house, it's nice to know where everything is and that items are placed in a logical order (hopefully your kitchenware isn't found in your bathroom).

Knowing where files are allows for consistency (even across multiple projects) and shareability.

File organization is a form of communication and requires you to practice some theory of mind.

Why file organization matters

Why do it: for others

It is always best to work under the mindset that your work will be shared and reproduced by others.

Organize your code such that newcomers can quickly access key parts of your project without looking too hard.

Why file organization matters

Why do it: for others

Practice theory of mind: Organize files so that newcomers thank you for:

- the time saved in learning your project structure
- being able to collaborate more easily with you
- quickly learning how your analysis was performed
- being able to quickly reproduce your code, which adds confidence to both the collaborator and you.

Why file organization matters

Why do it: for you

Time invested in learning how to organize your code now will help you streamline future projects and add consistency between projects.

Even if you are working on a quick homework assignment, think about the dozens of similar homework assignments for which you have to mock up a folder and then finagle some files.

Consistency and good habits decrease the amount of time it takes to boilerplate your code.

Why file organization matters

So you are not this clown

Why file organization matters

Why do it: for the future you

You are your own user!

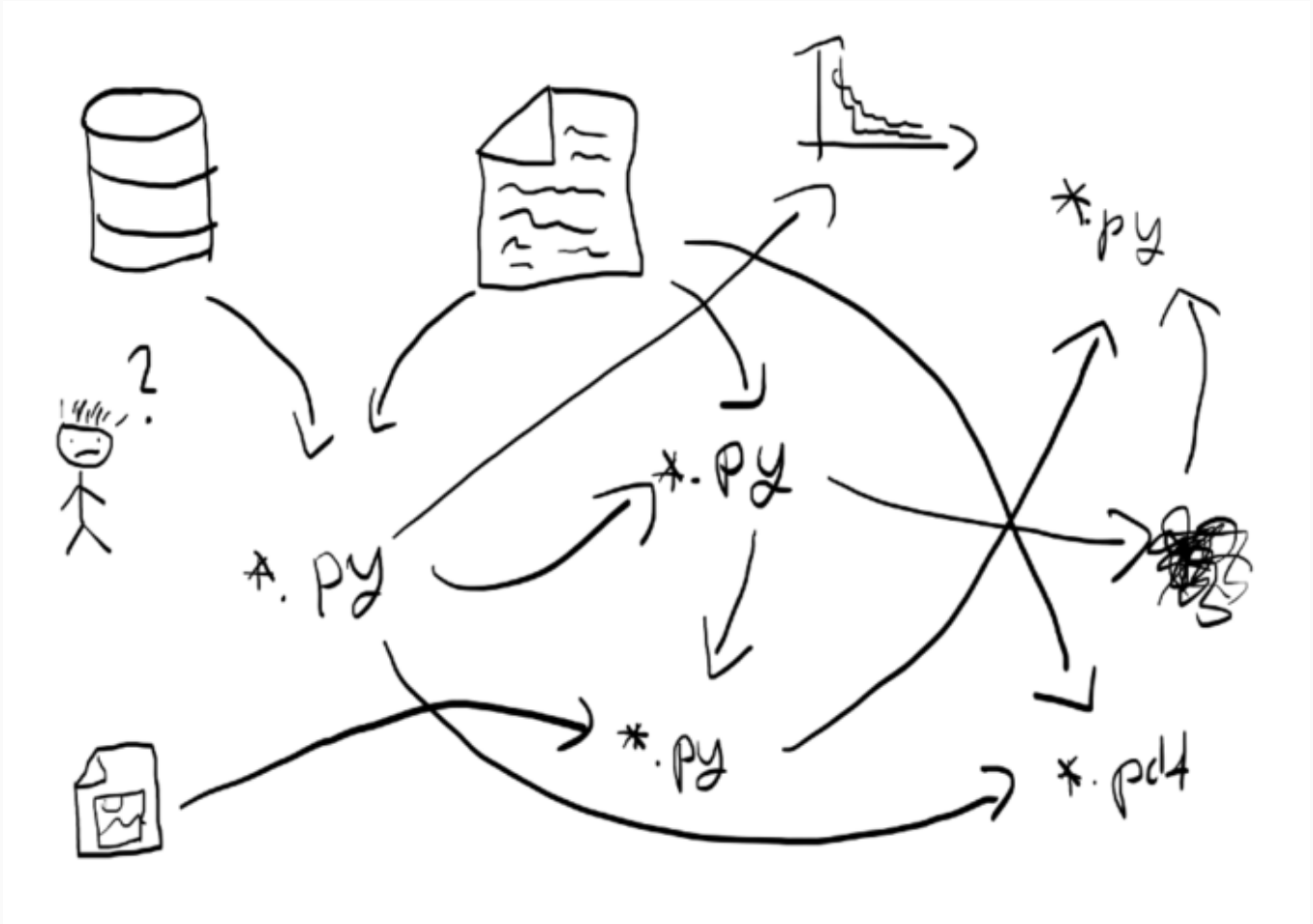
You don't want to have to back to previous code and have forgotten where your files are placed, or worse, which files to use (the deadly trial-and-error approach of using `file1.py`, `file-v1.py`, `file-mod1.py`, or `file-try-this-one-v1.py` suffixes).

It will limit the ability to reproduce your code.

Establishing consistency at the beginning – which starts with a standard file structure and naming convention – will pay dividends for your future self.

Why file organization matters

Why do it: for the future you



this sucks ^

Getting started

Some terminology:

- *Folders* and *directories* are used interchangeably.
- File structure, file hierarchy, and file schemas are used interchangeably.

Getting started

A state of mind: Not everything has to be set in stone.

None of these practices are binding, or necessary to make good code.

> *"A foolish consistency is the hobgoblin of little minds." — Ralph Waldo Emerson (and PEP8 style guides)*

A state of mind: Not everything has to be set in stone.

There are tradeoffs between the time to develop such file structures, and time to actually write and implement code.

> *"Premature optimization is the root of all evil (or at least most of it) in programming." — Donald Ervin Knuth*

Much like writing style, you should adapt good coding practice and data management habits to your specific needs while still being in sync with the rest of the world.

Best practices for file structures

There are some overarching ground rules to keep in mind before hacking away at your computer. They are:

1. Do not modify your raw data ever for any reason. Store them as read-only if you can.
2. Data manipulation should work like an assembly line: it stops at checkpoints. E.g., it gets modified/cleaned/analyzed, and then it moves on.
3. Code of different quality (scratch work vs. compiled binaries) should be separated.
4. Always have collaborators in mind, even if there is a 0% chance of getting collaborators. Work towards shareable code. Have public awareness. In a way, imagine if your code were to be released on GitHub right now.
5. **Consistency (within your project) is key.**

Best practices for file structures

Best practices for separation of concerns

When writing code, it is a good design principle to separate program functionality into distinct sections.

Each section of code should address a separate concern (e.g. reformatting data to plot a figure).

This is also true when breaking down your file structures: each folder should contain files with similar functionality or purposes.

Best practices for file structures

Best practices for separation of concerns

Data

- Data will be acted upon differently at multiple points along your project's assembly line.
- Separate out its evolution: Small projects get separate files, large projects get separate folders.
- Usually it's not worth having a folder with only one file in it.

Best practices for separation of concerns

Raw data

- What your participants, sensor, machine, experimental apparatus, etc. generate. **Raw data is immutable**, no matter how dirty, noisy, and ugly it is.
- Often bad file names come from data you get from other sources – a collaborator, the Internet. Keep these names, but write a script to transform the files into cleaner intermediate files. (That way, if you get more from the same source, or you share your analysis code with your collaborators, no one needs to manually repeat the work of transforming the data.)

Best practices for separation of concerns

Edited data

- Filtered or edited data is where you store the intermediate steps as the data is manipulated from a copy of the raw state to its final state.
- In social science, one first step is usually to de-identify the data (don't put PII on the internet!)

Best practices for file structures

Best practices for separation of concerns

Finalized data

- The final data is what will be accessed by the main analysis program, and used to derive the conclusions you wish to learn about.
- Separating this data prevents accidental misuse of intermediate or bad data that could potentially corrupt your results.
- And it helps clarify what's being used if your working version is not the highest version in your working data, while avoiding the ugly `_final_final` and `_this_one` filenames we're all too familiar with.

Best practices for file structures

Best practices for experimental and scratch work

To help keep track of this process, every scientist should have some sort of lab notebook or method of tracking data.

Definitely include	Maybe include
A date	Where did the data come from?
Comments on how the experiment was performed	Who did the experiment?
Observations about the output	Is this a repeat experiment, re-done with different parameters, modified datasets, or a new method?

Best practices for file structures

Best practices for experimental and scratch work

As you develop a project, scratch work will sprout up alongside your lab notebook. Scratch work is anything you do to transform, manipulate, or inspect (plot, explore, analyze, etc.) the data as it moves from station to station along the conveyor belt.

Always put everything into scripts to ensure repeatability, even though it may seem quick and easy to do with command line arguments in the moment.

Best practices for file structures

Best practices for experimental and scratch work

In research and industry, someone may always ask you to tweak some of your data manipulation, plotting, or analysis and you have to be able to recreate your work.

A good motto is "data to table" – have an automated pipeline that acts on the raw data, generates the edited data if needed, and produces all the tables and figures you need.

Best practices for file structures

Source code: Put it somewhere logical.

By source code, we mean the finalized working code that is used to derive the results.

Depending on the size and nature of your project, there may also be several single-use scripts for tasks such as converting, cleaning, plotting, and so on. Often scripts are for demonstration, so they can go in `demos`.

Whether these scripts are included in the `src` folder as final working code or distributed in other folders as scratch work is mostly a matter of personal preference, as long as it's logical to find them later. (I sometimes have a `scratch` folder.)

Best practices for file structures

Results: Keep them separate.

Whatever format your results are (figures, a code/executable, data to be passed onto another project, even a paper draft), keeping them separate allows quick access when you need to evaluate your conclusions.

Best practices for file structures

Consider using Makefiles.

Especially for larger projects, Makefiles will aid in tracking revisions in your code and data. (Google/LLMs and Makefile templates are your friends.)

File structures should match the scale and aims of the project

All good structures contain at least the following elements:

1. A unique main folder for the project
   - `/repo/projectName/`
2. Some form of code
   - `projectName/src/`
3. Some form of data
   - `projectName/data/`
4. A readme document with any important information about the project for yourself or collaborators
   - `projectName/README.md`
5. Probably a place to keep output files and one for documentation
   - `projectName/output/`
   - `projectName/docs/`

```
/repo/projectName/
        +-- data/
            +-- raw/
            +-- deidentified/
            +-- processed/
        +-- docs/
            +-- index.md
        +-- output/
        +-- src/
            +-- main.py
            +-- utils.py
            +-- preprocessing/
            +-- summaryStatistics/
            +-- modelBasedAnalysis/
        +-- README.md
```

File structures should match the scale and aims of the project

Start by sketching out a base structure.

1. Make a unique folder for the project.
2. Determine the scale of the project. Somewhere between quick visualization and long-term collaborative project? This will give a hint at how complex the file structure should be.
3. Identify the parameters distinguishing data. Someone looking at your files should be able to recognize those parameters when looking at both your file structure and filenames.
4. Assess the easiest way to access the data. Often, this will be set by the most important distinguishing variable between data sets (e.g., chronological, machine/sensor type, experiment, human subject, learning model, etc.). Keep things simple and clear to start, so it's easy to inspect and debug.

File structures should match the scale and aims of the project

Case studies: Different file structures are suited to different needs.

- Is this a small assignment that requires a quick output? See Case Study 1: almost flat.
- Is this an exploratory project, where there may not even be results given the uncertainty of the data and results? See Case Study 2: a simple hierarchy.
- Is this is a huge collaborative project which you expect to drive several distinct results sections, and possibly multiple manuscripts/publications? See Case Study 3: a complex hierarchy.

Best practices for naming conventions

It's easy to be overwhelmed by the sheer variety of naming conventions. However, there are a few important base principles on which you can build your own.

1. Be descriptive and avoid ambiguity, particularly with versioning (e.g., try `-v2`, not `_final_final`)
2. Keep names concise.
    - Abbreviations are helpful, but make sure you define them in a readme.
    - Use context (e.g. parent folders) to avoid redundant/lengthy names.

Best practices for naming conventions

3. There is only one acceptable way to format a date: `YYYY_MM_DD` (e.g. `2024_07_01`. Your operating system will automatically sort this style chronologically.
    - Similarly, always pre-pad smaller numbers with zeros in a sequence (e.g. 01, 02,...,10 if instead of 1, 2, 3,...,10).
4. As always, be consistent within your project. If your group already has an established style, start there, and tailor it to your needs.

Best practices for naming conventions

Construct a file path sentence that helps users identify the file.

Every file, whether it be data or code, has a path (including the filename at the end) that tells the computer where to look for it.

The path is also a "sentence" that tells the human user the information needed to identify the file. Separate this information into individual "idea elements."

Folder names are, in general, are a single idea element. Filenames, on the other hand, may be one or more ideas.

Best practices for naming conventions

Construct a file path sentence that helps users identify the file.

Take, for example, the raw data filename `participant01_rdmStudy-v2.1_2024_07_01.csv`. There are three idea elements here:

	Idea 1	Idea 2	Idea 3
General idea:	Participant ID	Experiment	Date recorded
Details:	`participant01`	`rdmStudy-v2.1`	`2024_07_01`

Best practices for naming conventions

Ordering the ideas: Choosing descriptive vs. chronological depends on what's most important.

How you choose to order the qualifiers in a file path sentence (should it be date, then location – or location, then date?) depends on how you will be accessing the data.

In general, put the most important thing first. If the project goal is to identify temporal variations in data, put the date first in the path sentence (left-most, highest-level).

Often it makes most sense to organize by experiment first, then maybe by participant group. Organizing by date makes sense for a longitudinal study but less so for a cross-sectional one.

Best practices for naming conventions

Ordering the ideas: Choosing descriptive vs. chronological depends on what's most important.

The last idea to be addressed is often the version number. Use a simple version sequence (e.g. `_v1`, `_v2`, `_v3`) to avoid the chaos of `_final`, `_finalfinal`, `_reallyfinal`, `_test_final_use_this_one`.

If someone handed you files like this, would you trust the conclusions? Would you feel they did a good job with the project?

Best practices for naming conventions

Separating the ideas: Underscores, camelCase.

To prevent ideas from blurring together, some form of delimiter is needed. The OS already put slashes in between folder names for us, but what if we have multi-word ideas or multi-idea filenames?

`snake_case` and `camelCase` are probably the two most common delimiters. They are robust and unlikely to give you trouble when sharing your project.

The rule of camelCase is that *only* the first letter of subsequent words is upper case, everything else is lowercase (even if that would normally be uppercase, like `firstName` or `rdmData`).

I often use snake_case for objects that perform actions (`process_data.py`) and camelCase for modifiers (`processedData.csv`), but that's just me.

Best practices for naming conventions

Separating the ideas: Underscores, camelCase.

Mixing delimiters can be a great strategy for improving readability. Consider these two filenames:

- `20190701.GeorgeSun.v1.csv`
- `20190701_davidLarson_v1.csv`

The general ideas are are date, subject name, and version number. In both cases, a primary deliminator is used to separate the ideas, while camelCase is used to make the names easier to read.

Best practices for naming conventions

Separating the ideas: Underscores, camelCase.

A last note on delimiters: **never use spaces in your project path and filenames**.

Yes, many of us use them in our computer's general file system because they're so intuitive.

But code and operating systems can have a hard time dealing with them, so avoiding them in your project file structures will save you a lot of headache and cumbersome code.

References

The content of this lecture is substantially based on File Structure, published by the MIT Communication Lab.

Version Control III

Joachim Vandekerckhove

Winter 2025