Version Control for Data

Contents

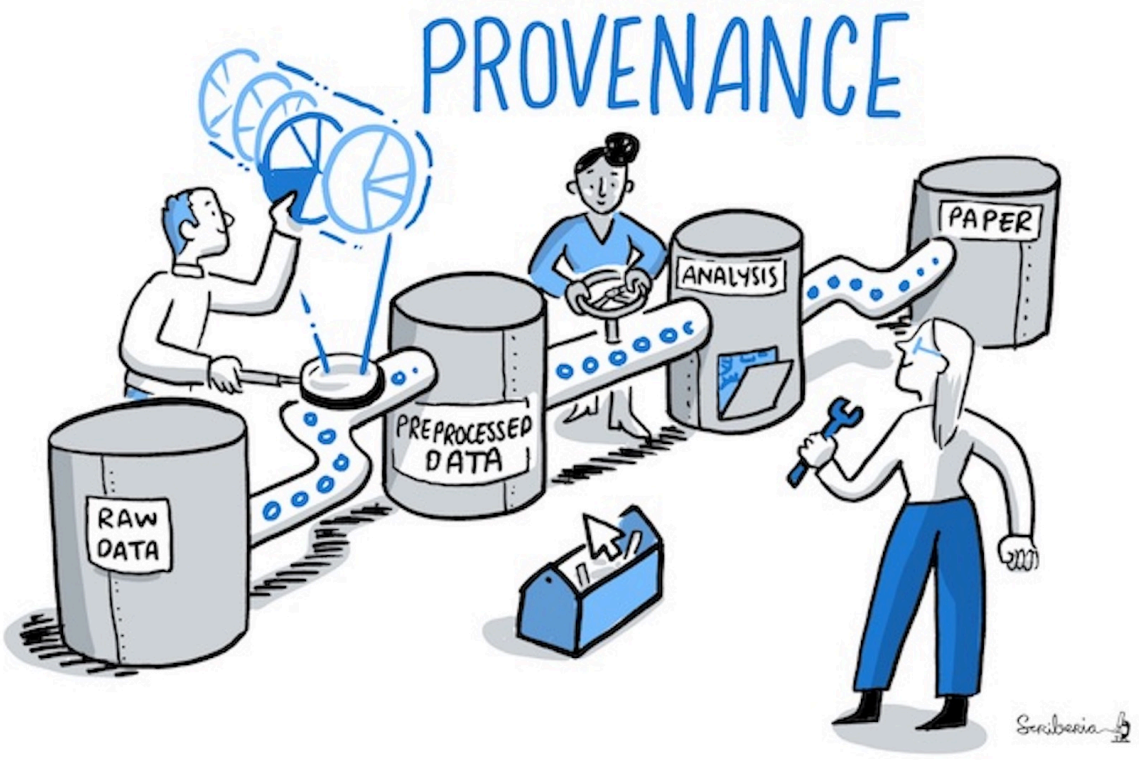

Fig. 34 Provenance on which data in which version was underlying which computation is crucial for reproducibility. *The Turing Way* project illustration by Scriberia. Used under a CC-BY 4.0 licence. DOI: 10.5281/zenodo.3332807.

We discussed that version controlling the components of evolving projects could help to make work more organised, efficient, collaborative, and reproducible. Many scientific projects, however, do not only contain code, manuscripts, or other small-sized files, but contain larger files such as large datasets, analysis results, or binary files (presentations,

manuscripts, pdfs) which can change or be updated in a project just like other small sized text components. In this chapter, we discuss why and how to do data versioning, especially why Git is not well suited for data versioning and what we can be done about it.

Importance of Version Controlling Data

We should not hold the notion that the data used for analysis is static; once it is acquired, it does not change and serves as input for a given analysis and the backbone of our scientific results. The reality is that data is only rarely invariant. For example, throughout a scientific project, datasets can be extended with new data, adapted to new naming schemes, reorganised into different file hierarchies, updated with new data points or modified to fix any errors. Sometimes you might also want to experiment off different versions of the same dataset.

Such dynamic processes are excellent and beneficial for science as they ensure that data is usable and up-to-date, but they can be confusing if they are not adequately documented. If a dataset that is the basis for computing a scientific result changes without version control, reproducibility can be threatened: results may become invalid, or scripts that are based on file names that change between versions can break. Especially if original data gets replaced with new data without version control in place, the original results of the analysis may not be reproduced. Therefore, version controlling data and other large files in a similar way to version controlling code or manuscripts can help ensure the reproducibility of a project and capture the provenance of results; that is "the precise subset and version of data a set of result originates from". Together with all other components of a research project, data identified in precise versions is part of the research outcome. The reproducibility aspect of a scientific project can improve a lot if we can track the subset or version of data a certain analysis or result is based upon.

Challenges in Version Controlling Data

As we described earlier, there are limitation to git. As long as the files to version control are small in size, not too numerous and can be stored in a few `csv` or character separated files, tools such as Git are appropriate.

However, when you work, share, and collaborate on large, potentially binary files (such as many scientific data formats), you need to think about ways to version control this data with specialised tools. If others try to clone your repository or fetch/pull to update it locally,

it will take longer to do this if it contains larger files that have been versioned and modified.

Accordingly, repository hosting services usually impose maximum file sizes on users. For example, if a single file in your repository exceeds 100MB, you will not be able to push this file to a GitHub repository. Furthermore, if a large file was accidentally added to a repository, removing the file from the repository can be tedious, as this file needs to be purged.

These shortcomings can make version controlling files tedious and slow, impede collaborations on repositories with large data, and prevent data or projects with data from being shared on platforms like GitHub.

Tools for Version Controlling Data

Several tools are available to handle version controlling and sharing large files. Most of them integrate very well with Git and extend a repository's capabilities to version control large files. With these tools, large data can be added to a repository, version controlled, reverted to previous states, or updated and modified collaboratively, and even shared via GitHub as small-sized files. Some of these tools include:

DVC

DVC (open-source Version Control System for Machine Learning Projects) https://dvc.org/. DVC guarantees reproducibility by consistently maintaining a combination of input data, configuration, and the code that was initially used to run an experiment.

Git LFS

Git LFS comes with a command-line extension to Git and allows you to treat files of any size alike, using standard Git commands. A major shortcoming, however, is that Git LFS is a *centralised* solution. Large files are not distributed but stored on a remote server. This usually requires setting up your server or paying for a service - which can make it very inaccessible.

`git-annex`

The `git-annex` tool is a distributed system that can manage and share large files independent from a central service or server. `git-annex` manages all file *content* in a separate directory in the repository (`.git/annex/objects`, the so-called *annex*) and only places file *names* with some metadata into version control by Git. When a Git repository with an annex is pushed to a web-hosting service such as GitHub, the contents stored in the annex are not uploaded. Instead, they can be pushed to a storage system (such as a web server, but also third party services such as Dropbox, Google Drive, Amazon S3, box.com, and many more). If a repository with an annex is cloned, the clone will not contain the *contents* of all annexed files by default, but display only file names. This makes the repository small, even if it tracks hundreds of gigabytes of data, and cloning fast, while file contents are stored in one or more free or commercial external storage solutions. On-demand, any file content can then be obtained with a `git-annex get` command from the external file storage.

git submodules

Submodules allows to split the data in different repositories, while keeping everything under a single "parent" repository. It is very powerful, but difficult to use. Especially, using Git Branches in submodules make it complex to handle. However, this is the only tool listed here allowing to work with many files in a Git repository.

DataLad

DataLad, builds upon git and git-annex. Like `git-annex`, it allows you to version control data and share it via third-party providers but simplifies and extends this functionality. In addition to sharing and version controlling large files; it allows recording, sharing, and using software environments, recording and re-executing commands or data analyses, and operating seamlessly across a hierarchy of repositories.

Data versioning and inclusivity

Data versioning in Git require the use of more complex tools, and this means that accessibility to the data will be more difficult. For instance, if you use datalad with Github, newcomers trying to see one of the large file will have difficulties: they will be able to see that the file exists, but will not be able to download or see it without cloning the repository and running git-annex or datalad commands.

So while using these tools will make Git commands to run faster, one may want to disable them for critical binary files, like presentations or pdfs. A solution can be to pack them in submodules, so that the repositories are keeping a small size.

As an example, we can take the repository creating the turing book. The repository is slow to work with, because a lot of binary files were used over the time. However, it makes the onboarding of new users easier.