PP30

Snakemake for Statistical Programmers: Automation and Reproducibility in Clinical Studies

Mathieu Cayssol Nassim Sleiman

Hoffmann–La Roche, Basel, Switzerland

Abstract

Snakemake is a workflow management system that automates complex data pipelines by specifying rules and dependencies. It ensures reproducibility, optimizes runtime through parallel execution, and minimizes manual intervention. At Roche, we have developed a standardized Snakemake pipeline (integrating essential dependencies) to automate the creation of SDTMv, ADaMs, and TLG outputs using SAS and R. Designed specifically for statistical programmers, this approach streamlines the process of generating clinical datasets and analytical reports, improving transparency and consistency. By leveraging Snakemake's capabilities, statistical programmers can reduce processing time and maintain robust, traceable workflows. This poster will highlight our pipeline structure and the transformative impact of adopting Snakemake for clinical data reporting tasks.

1 Introduction

Statistical programmers operate under tight timelines and strict compliance. Sponsors and regulators expect evidence of reproducibility, selective re-runs when data change, and full auditability. Workloads span SAS and R, with frequent late updates to specifications and data. The result is pressure to deliver fast while keeping every step transparent and traceable. The challenge is practical: how do we ensure that only the right tasks are re-run, that outputs are consistent across environments, and that every result can be traced back to code, inputs, and versions? Ad-hoc scripts and manual orchestration struggle here. They invite errors, make impact analysis slow, and leave gaps in the audit trail. A workflow management system addresses these risks. We selected Snakemake because it is lightweight and uses simple, declarative rules. It builds a dependency graph, runs jobs in parallel, and re-executes only what is affected by a change. It also integrates cleanly with both SAS and R, enabling a unified approach without heavy infrastructure. In this work, we present a reusable template Snakemake workflow for clinical reporting. It standardizes project structure, configuration, and logging. It includes ready-to-adapt rules for SDTM, ADaM, and TLG generation, with built-in provenance and run metadata. The template shortens onboarding, reduces rework, and strengthens validation evidence.

2 Clinical Study Workflow

The CDISC 360i Program focuses on enabling standards-driven automation from study design through results. The program aims to realize the CDISC vision of amplifying data's impact by creating connected, digitized standards to reduce variability and increase automation. This figure shows the entire clinical study lifecycle, broken down into four main phases: **Design**, **Build**, **Run**, and **Adapt**.

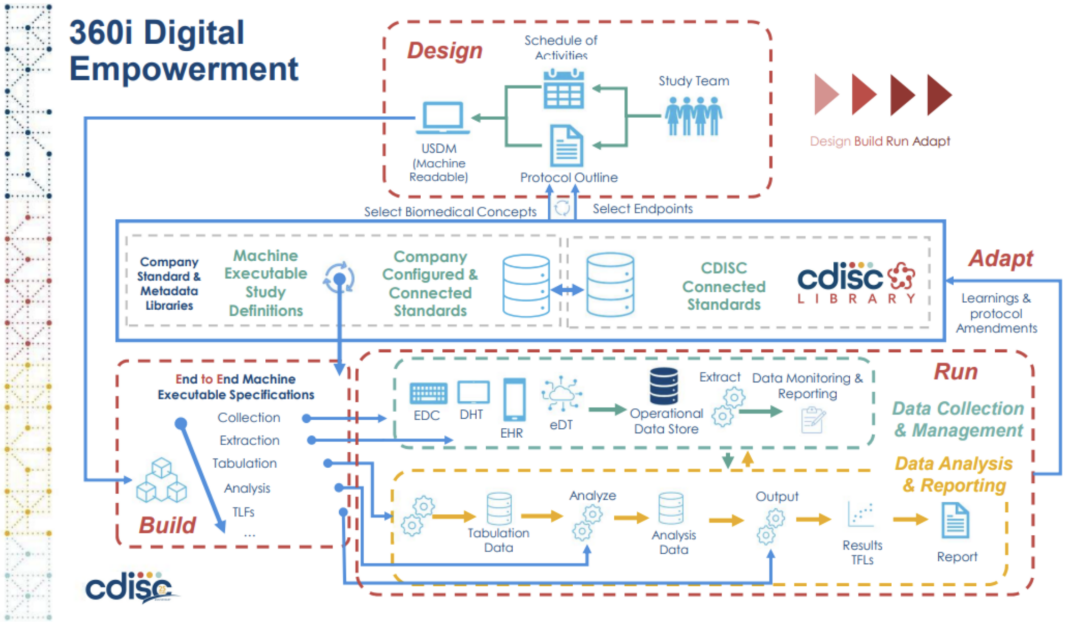


Figure 1: Clinical study lifecycle spanning Design, Build, Run, and Adapt.

Design: The study begins here. The study team uses a protocol outline to identify key endpoints and biomedical concepts. They then generate a machine-readable study definition using the **USDM** (Unified Study Data Model), which serves as a digital blueprint for the entire study.

Build: The digital study specifications are developed in this phase. This involves translating the concepts from the Design phase into machine-executable specifications for data collection, tabulation, analysis, and **TFLs** (tables, figures, and listings). These specifications are built using company standard libraries and aligned with **CDISC** standards.

Run: The execution phase, where data is collected, managed, and analyzed. Data originates from multiple sources, such as **EDC** (Electronic Data Capture), **EHR** (Electronic Health Records), and **DHT** (Digital Health Technologies). It is extracted and stored in an Operational Data Store, then processed into Tabulation Data and Analysis Data. These datasets are used to generate the final Results **TFLs** and the study Report.

Adapt: The final phase creates a feedback loop. Insights from the study results and analysis are used to refine and amend the protocol, feeding back into the Design phase to improve future studies.

The following figure provides a closer look at the Run phase for Data Analysis & Reporting. At Roche, we use Snakemake during this phase to produce tabulation and analysis datasets as

well as TFLs, ensuring that executions are auditable, efficient, and reproducible.

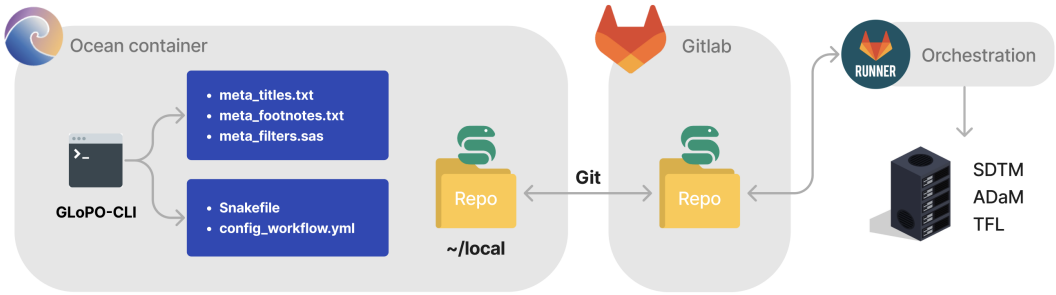


Figure 2: Run Phase Process: Data Analysis & Reporting at Roche

GLoPO-CLI: A command-line tool designed to extract content from the GLoPO Spreadsheet and generate key metafiles required for study activities. It produces files such as `meta_titles.txt`, `meta_footnotes.txt`, `meta_filters.sas`, GitLab issues (XLSX), and Snakemake workflow files (e.g., `workflow_lopo.smk`, `Snakefile`, `config_workflow.yml`). It automates metadata extraction and generates customized Snakemake workflow files tailored to the specific study.

GitLab Repository: The GitLab repository stores all code and metadata associated with a specific analysis activity. It acts as the single source of truth, ensuring version control and traceability. Programmers typically work on a local clone of this repository.

Metadata Files: The analysis is guided by a set of metadata files that separate analytical content from the underlying code. Examples include: `meta_titles.txt` and `meta_footnotes.txt` for managing TFL titles and footnotes. `meta_filters.sas` for enabling TFL production across different subsets of patients or data.

Snakemake Workflow: The Snakemake workflow management system forms the core of the automation. The `Snakefile` and `config_workflow.yml` define the rules, dependencies, and configuration required to generate the final outputs.

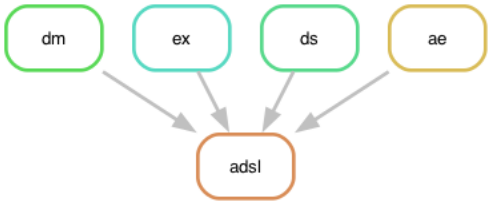
Orchestration: Once local changes are finalized, they are committed and pushed to the remote GitLab repository. A GitLab runner then orchestrates the Snakemake workflow, ensuring that SDTM, ADaM, and TFL outputs are produced in a consistent, reproducible, and auditable manner.

3 Snakemake

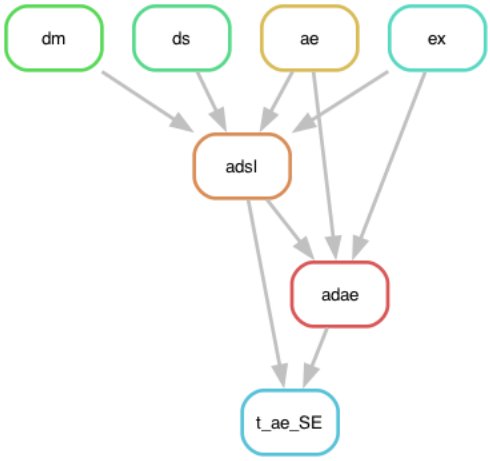
Snakemake is a workflow management system for creating **reproducible, scalable data analyses**, described in a readable, Python-based language. A single workflow definition can run on a laptop and scale to servers, clusters, grids, or cloud environments without changing the workflow code. Analyses are specified as **rules in a Snakefile** that map outputs to inputs. A shell command is specified to generate the desired outputs from the given inputs. From these rules, Snakemake infers a **directed acyclic graph (DAG)** of jobs and executes it with automatic parallelization. It also performs **incremental builds**, rerunning steps only when inputs are newer than outputs. Below is an example of Snakemake rules for SDTM and ADaM datasets and TLG.

```
# SDTM rules
rule dm:
    output: "data/sdtm/dm.parquet"
    shell: "Rscript programs/sdtm/create_dm.R"

rule ex:
    output: "data/sdtm/ex.parquet"
    shell: "Rscript programs/sdtm/create_ex.R"

rule ds:
    output: "data/sdtm/ds.parquet"
    shell: "Rscript programs/sdtm/create_ds.R"

rule ae:
    output: "data/sdtm/ae.parquet"
    shell: "Rscript programs/sdtm/create_ae.R"

# ADAM rules
rule adsl:
    input:
        dm="data/sdtm/dm.parquet",
        ex="data/sdtm/ex.parquet",
        ds="data/sdtm/ds.parquet",
        ae="data/sdtm/ae.parquet"
    output: "data/adam/adsl.parquet"
    shell: "Rscript programs/adam/ad_adsl.R"

rule adae:
    input:
        ae="data/sdtm/ae.parquet",
        ex="data/sdtm/ex.parquet",
        adsl="data/adam/adsl.parquet"
    output: "data/adam/adae.parquet"
    shell: "Rscript programs/adam/ad_adae.R"

# TLG rules
rule t_dm_SE:
    input: "data/adam/adsl.parquet"
    output: "data/tlg/t_dm_SE.out"
    shell: "Rscript programs/sdtm/t_dm.R --filter SE"

rule t_ae_SE:
    input:
        adsl="data/adam/adsl.parquet",
        adae="data/adam/adae.parquet"
    output: "data/tlg/t_ae_SE.out"
    shell: "Rscript programs/sdtm/t_ae.R --filter SE"

# All SDTM, ADAM and TLG rules
rule all_sdtm:
    input:
        "data/sdtm/dm.parquet",
        "data/sdtm/ex.parquet",
        "data/sdtm/ds.parquet",
        "data/sdtm/ae.parquet"

rule all_adam:
    input:
        "data/adam/adsl.parquet",
        "data/adam/adae.parquet"

rule all_tlg:
    input:
        "data/tlg/t_dm_SE.out",
        "data/tlg/t_ae_SE.out"
```

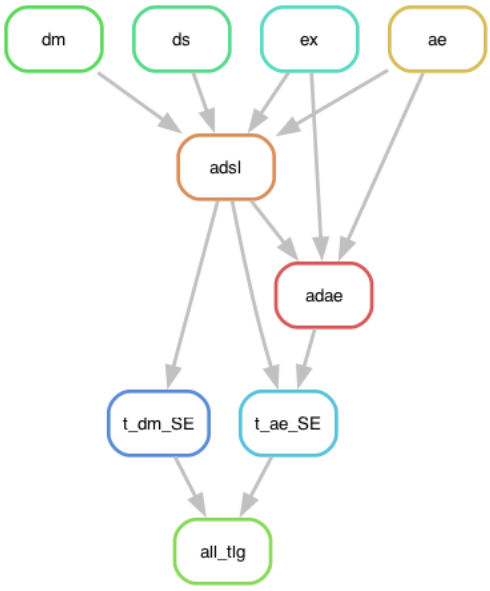
Snakefile



DAG generated with adsl as the target rule



DAG generated with t_ae_SE as the target rule



DAG generated with all_tlg as the target rule

Figure 3: Example of Snakefile and its corresponding DAGs

Execution is primarily via the command-line interface. The same workflow can target local cores or distributed backends through executor plugins (e.g., Slurm, generic cluster, Kubernetes). Given the previous snakefile, we can use the Snakemake CLI executing the following commands:

```
# Generate only 2 SDTM datasets (dm, ae) - using 2 cores for parallelization
snakemake -j2 dm ae

# Generate a specific ADaM dataset (e.g., adsl) - using 4 cores for parallelization
snakemake -j4 adsl

# Force regeneration of all the ADaM datasets and the upstream dependencies
# using 4 cores for parallelization
snakemake -j4 --forceall all_adam

# Dry-run: show what would be generated (no actual execution)
snakemake -j4 --dry-run
```

Figure 4: Example Snakemake Commands

For reproducibility and portability, Snakemake integrates software deployment per rule using containers; making the software stack an explicit, versioned part of the workflow.

```
# Rule to produce adae in a public R container
rule adae:
    container:
        "docker://rocker/r-base:4.2.0" # Example public R container from Docker Hub
    input:
        ae="data/sdtm/ae.parquet",
        ex="data/sdtm/ex.parquet",
        adsl="data/adam/adsl.parquet"
    output:
        "data/adam/adae.parquet"
    shell:
        "Rscript programs/adam/ad_adae.R"
```

Figure 5: Example of the container field in a Snakemake rule

Workflows support modularity and reuse via importable modules and a maintained wrapper repository.

```
# Module named workflow_sdtm containing all the SDTM rules
module workflow_sdtm:
    snakefile:
        "workflow_sdtm.smk"

# Import all the sdtm rules
use rule * from workflow_sdtm

# Module named workflow_sdtm containing all the ADAM rules
module workflow_adam:
    snakefile:
        "workflow_adam.smk"

# Import only the adsl rule
use rule adsl from workflow_adam
```

Figure 6: Modules and imports in Snakemake

The advantages of using Snakemake are the following:

Reproducibility: Ensures that both dataset creation and TLG generation can be precisely reproduced, supporting audit trails and regulatory requirements. Snakemake adheres to compliance and regulatory standards in clinical reporting.

Scalability: Handles everything from small pilot studies to large, multicenter trials, running seamlessly on desktops, containers, HPC clusters, or the cloud, without modifying workflows.

Automatic Job Scheduling & Dependency Management: Ensures that each data processing or analysis step (e.g., raw data cleaning, SDTM, ADaMs, TLG generation) runs in the correct order and in parallel when possible, reducing manual errors and accelerating execution.

Ease of Use & Modularity: Clear, Python-based syntax and a modular structure make it straightforward to maintain and update pipelines as study requirements or standards evolve.

Environment & Container Support: Built-in support for Conda environments and containers ensures consistent software versions and dependencies across different computing environments, which is key for regulatory compliance and collaboration.

4 Implementation at Roche

At Roche, the Snakemake workflow is designed to streamline clinical data processing by systematically defining rules for each step. Specifically, one rule is defined for every SDTM domain, one rule for each ADaM domain, and one rule per TLG. The Snakefile is distributed together with a configuration file, `config_workflow.yml`, which is extracted at the same time. The Snakefile reads this configuration file and adapts its rules accordingly, allowing statistical programmers to modify dependencies and study parameters without directly editing the workflow code. This separation between workflow logic (Snakefile) and study-specific settings (configuration file) ensures both flexibility and reproducibility. By default, the workflow assumes the following set of dependencies:

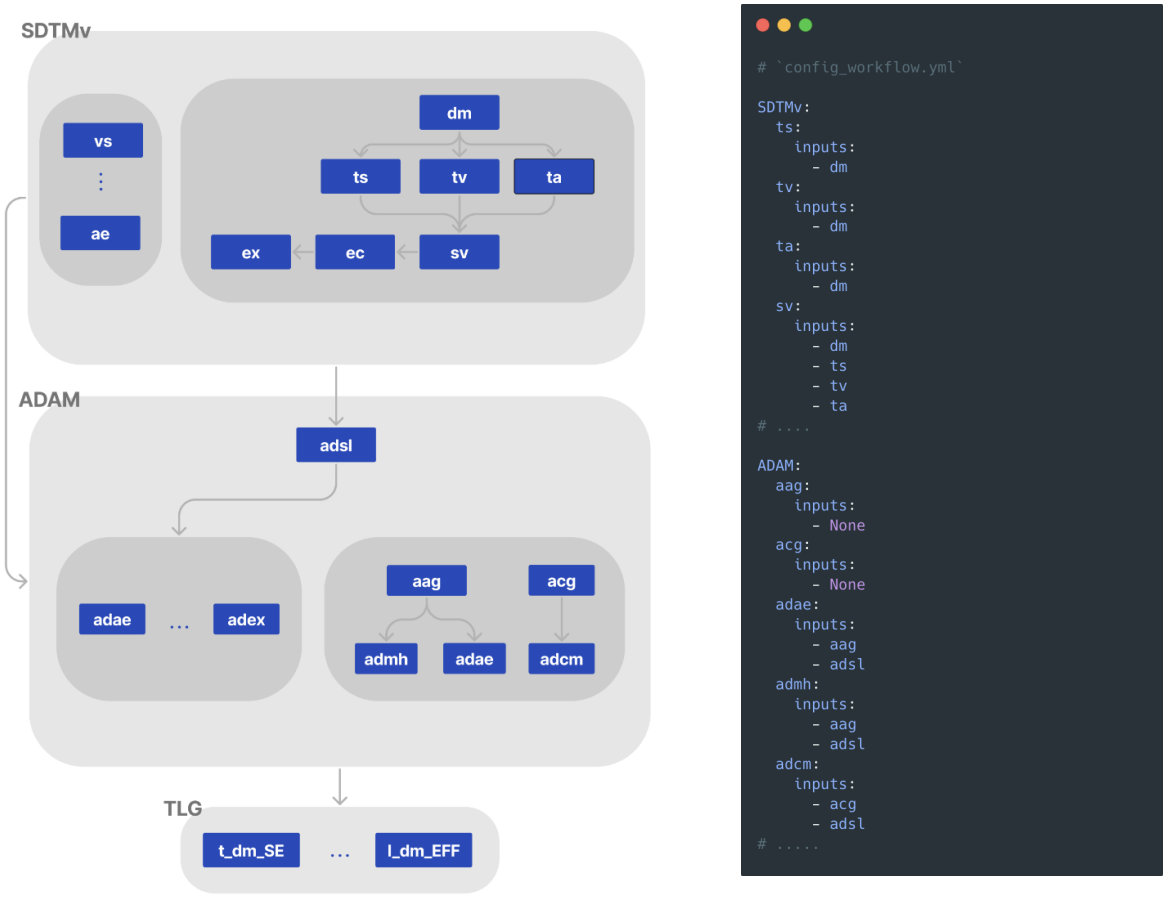


Figure 7: Default set of dependencies defined in `config_workflow.yml`

The configuration file provides a concise representation of dataset dependencies, which can be reviewed and adapted by the user before executing the workflow.

SDTM Dependencies

- TS, TV, and TA depend on DM.
- SV depends on DM, TS, TV, and TA.

ADaM Dependencies

- AAG and ACG serve as starting datasets (no inputs required).
- ADAE and ADMH depend on AAG and ADSL.
- ADCM depends on ACG and ADSL.

These dependencies are automatically inferred by Snakemake based on the input and output files defined in the Snakefile rules. Beyond modifying the default dependencies, users can further customize their study by parametrizing the workflow through additional options in the configuration file:

```yaml
# data and program paths
paths:
  sdtmv:
    program: "programs/sdtmv"
    data: "data/sdtmv"
    log: "data/log"
    overwrite: True
  adam:
    program: "programs/adam"
    data: "data/adam"
    esub: "inst/extdata/esub_adam"
    log: "data/log"
    overwrite: True
  tlg:
    program: "programs/tlg"
    data: "data/output"
    log: "data/log"

# study configuration sdtm and datacut
study_config:
  cutoffdate: ""
  sdtm:
    enable: False
    directory: "data/esub_sdtm"
    program: "programs/sdtmv/create_sdtm.R"
    esub: "inst/extdata/esub_sdtm"

  datacut:
    enable: False
    directory: "data/sdtmvcut"
    program: "programs/sdtmv/create_datacut.R"
```

Figure 8: Study-specific configuration options in `config_workflow.yml`

This configuration section is used to define various options, data paths, and program paths. It enables users to activate datacut and/or SDTM creation, define dataset directories, and specify program names. The datacut option determines whether a clinical cutoff date should be applied to the datasets, while SDTM creation refers to the generation of SDTM datasets based on the Roche standard SDTM model (SDTMv). The section also provides an option to set a study cutoff date. Because these options are read directly by the Snakefile, any changes made by statistical programmers are automatically reflected in the executed workflow.

5 Challenges

While Snakemake provides significant benefits in terms of automation, reproducibility, and scalability, several challenges remain when applying it to clinical reporting workflows:

Challenge 1 - Flexibility vs. Traceability: Even a minor change in a dataset often results in rerunning all downstream tasks. This behavior ensures full traceability, since every output is guaranteed to be aligned with the most recent inputs. However, the lack of flexibility can make releasing a study cumbersome. For instance, if a small correction is made to a demographic variable in the `DM` domain, the workflow may trigger regeneration of all downstream ADaM datasets and TLGs, even when most of them are unaffected. While this ensures audit readiness, it also increases computational costs and delays turnaround times.

Challenge 2 - Centralized Sources for All Targets: When a single file (such as titles or footnotes) serves as the source for all ADaM datasets or TLGs, even a small update can cascade into a full refresh of all outputs. For example, modifying just one footnote in `meta_footnotes.txt` may require rerunning the whole set of planned TLGs. This centralized approach ensures consistency across deliverables but creates inefficiencies when minor edits lead to full-scale re-execution of the workflow.

Challenge 3 - Manual Dependencies & Edge Cases: Even with a standardized workflow, edge cases frequently arise across studies that require manual intervention. For example, certain studies may have custom derivations for adverse event datasets (`ADAE`) or rely on non-standard external data sources that fall outside the default dependency structure. In such cases, programmers need to override or extend the predefined workflow, introducing additional complexity. These manual adjustments can increase maintenance burden and reduce the efficiency gains expected from automation.

Challenge 4 - Abstraction vs. Transparency: When workflow logic is heavily abstracted through Python code and controlled by a separate configuration file, the underlying processes can become opaque. For example, if an error occurs during execution, its root cause may be difficult to trace. The error might not originate from a clear, static rule in the workflow definition but from a dynamic dependency created by a specific parameter in the configuration file. While this approach offers flexibility by separating configuration from logic, it complicates debugging and makes the workflow harder to interpret. This lack of transparency can slow down troubleshooting and obscure critical details needed for full traceability.

Taken together, these challenges highlight the trade-off between reproducibility and practical usability. While the standardized Snakemake workflow strengthens transparency and auditability, further improvements are needed to handle study-specific variations more flexibly and to avoid unnecessary re-execution for minor updates.

6 Future implementation

Our future implementation will directly target the challenges of manual dependency management and workflow abstraction (challenges 3 and 4). To achieve this, we will move away from the current interdependent structure and adopt a new approach. Instead of dynamically linking the `Snakefile` to the configuration file (`config_workflow.yml`), dependencies and study settings will be defined through a dedicated web application named Atlas. This interface will allow statistical programmers to specify study metadata, dataset dependencies, and analysis options in an intuitive way. Once defined, the system will generate a fully explicit `Snakefile`, containing rules written out in clear form without hidden Python abstractions.

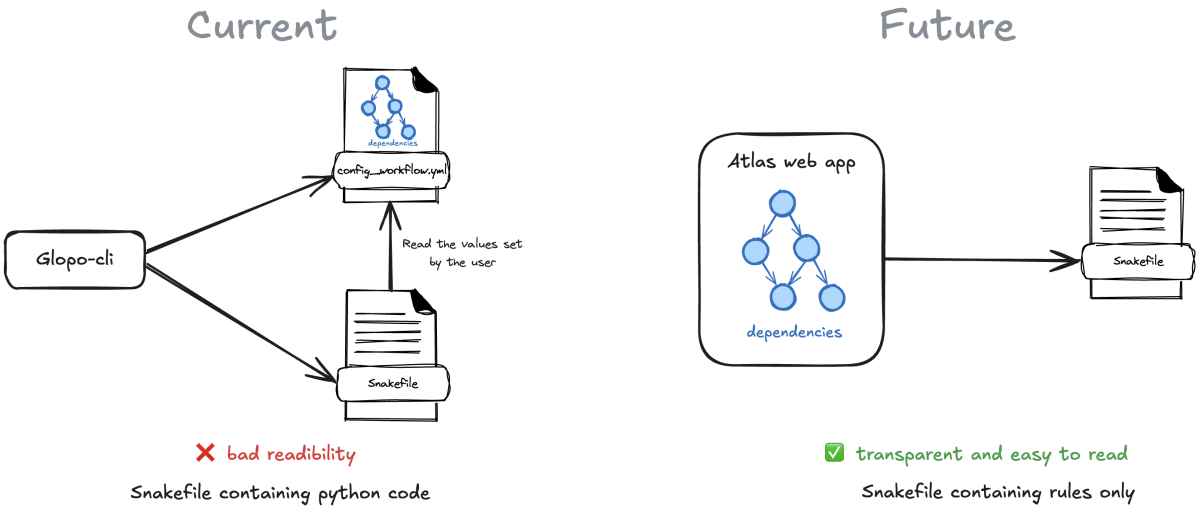


Figure 9: Current vs. Future Implementation of Snakefile Generation

This change will bring several advantages:

- **Improved Transparency:** Every rule and dependency will be directly visible in the generated `Snakefile`, making workflows easier to understand and audit.
- **Simplified Debugging:** Errors can be traced directly to explicit rules, avoiding the indirection caused by Python abstractions.
- **Study-Specific Flexibility:** The web-based configuration ensures that unique requirements and edge cases can be handled at the definition stage, without complicating the underlying workflow code.
- **Consistency Across Studies:** By standardizing how dependencies and configurations are captured upfront, programmers can maintain reproducibility while still benefiting from explicit workflow definitions.

This evolution transforms the workflow into a more transparent, user-friendly, and audit-ready system while preserving Snakemake's strengths in reproducibility, scalability, and automation.

7 Conclusion

The adoption of Snakemake in clinical data reporting at Roche highlights the value of workflow management systems in transforming statistical programming practices. By automating the generation of SDTM, ADaM, and TLG outputs, Snakemake ensures reproducibility, reduces manual intervention, and provides a transparent audit trail that meets regulatory requirements. Its integration with SAS and R, together with a declarative rule-based structure, enables efficient handling of complex dependencies and seamless scaling from local machines to high-performance computing environments.

The current implementation has already delivered key benefits:

- **Efficiency** through parallel execution and selective re-runs
- **Reproducibility and transparency** supporting audit and compliance
- **Standardization** of workflows that reduce onboarding and rework

Challenges remain, particularly in balancing flexibility with traceability, limiting unnecessary re-execution, and handling study-specific edge cases. The planned shift to explicit Snakefile generation through a web-based configuration will improve transparency, simplify debugging, and better accommodate unique study requirements.

In conclusion, Snakemake provides a solid foundation for automation and reproducibility in clinical reporting. Its continued evolution will further streamline workflows, strengthen compliance, and allow statistical programmers to focus more on scientific insight than process orchestration.

Acknowledgments

We would like to express our gratitude to the GLoPO-CLI team: Daphne Grasselly, Craig Gower-Page, Mahdi About and Helene Royo for their contributions. Our thanks also go to the Atlas team, including Daphne Grasselly, Haoyang Ju, Franciszek Walkowiak, Dinakar Kulkarni, Eddy Foster, and Rohan Parmar, for their dedicated efforts. Special thanks to the onboarding team on OCEAN: Helene Royo, Christof Binder, Valeria Visona, and Ivan Robinson for their support and collaboration.

Contact Information

mathieu.cayssol@roche.com
nassim.sleiman@roche.com

References

[1] Snakemake documentation: https://snakemake.github.io

[2] CDISC 360i Program: https://www.cdisc.org/sites/default/files/pdf/360i-Program-Kickoff2025.pdf

[3] Köster, J., & Rahmann, S. (2012). Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19), 2520–2522. Available at: https://doi.org/10.1093/bioinformatics/bts480

[4] Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., et al. (2021). Sustainable data analysis with Snakemake. *F1000Research*, 10, 33. Available at: https://doi.org/10.12688/f1000research.29032.2

[5] Köster, J., et al. The Snakemake workflow management system documentation. Retrieved October 7, 2025, from https://snakemake.readthedocs.io/