Forge: A Modern Build System

Technical Report

Shiladitya De

April 28, 2025

Forge is a modern, user-friendly build system designed to simplify and optimize the software build process. Developed in *C++20*, Forge focuses on providing a flexible, efficient, and highly readable framework suitable for projects of any size. Leveraging advanced language features, modular design principles, and modern development tools, Forge ensures that the build experience is both powerful and accessible. For example:

1 Key Features

1.1 Readable and Modular Build Configurations

Forge supports the decomposition of build files into smaller, reusable configurations. Similar to Python modules, this allows for better organization and maintainability, especially in large projects.

Forge Build Script Example

```
translate = import "translator.forge"
cl = import "clean.forge"

run: translate.test_res
    ./test_res inp.txt > res.txt

clean: cl.clean
```

1.2 Python-Like Imports

The import system in Forge mirrors Python's import mechanisms, providing intuitive syntax for including external configuration files and avoiding redundant parsing efforts. Moreover, it imports only those variables/targets which are needed in the current build, thus optimizing the process. It is different from the original *include* statement is make which works like *#include* in C.

1.3 HTML Build Outputs

Upon building, Forge can generate detailed HTML reports. These visually structured outputs improve clarity, helping developers quickly identify issues and monitor build statuses. It displays time taken by each build target in a gantt-chart (even in case of multiple jobs running).

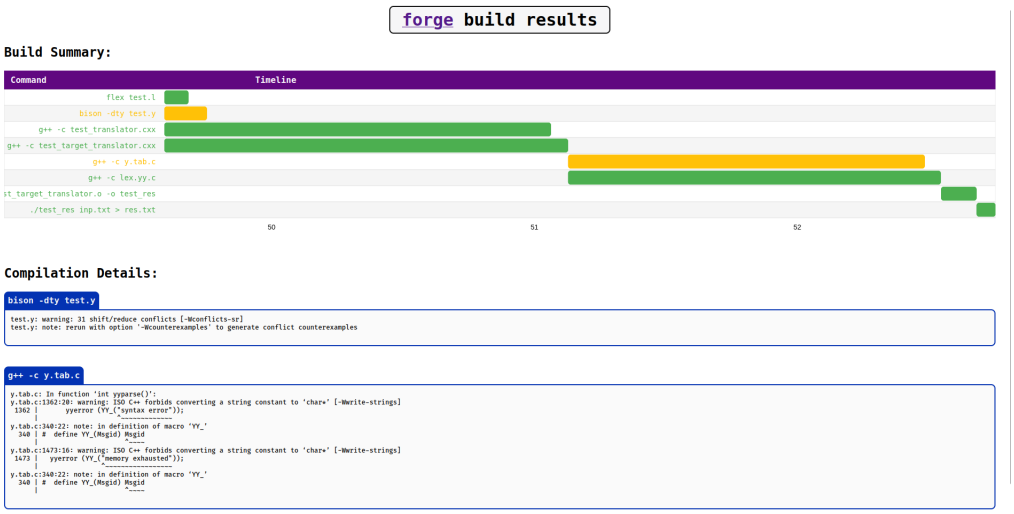


Figure 1: HTML Build Output

1.4 Caching Mechanism

Forge uses an internal caching system based on *SQLite3* databases. Forge stores into its cache the files which are already built. Thus, it optimizes the build by only rebuilding the libraries which are changed. Internally, it stores SHA256 hash of the file. Thus, it is safe to use in distributed environments as well where the concept of time is little tricky.

1.5 Parallel Processing

Task execution is parallelized wherever possible, utilizing multicore CPUs to accelerate the build process. Based on the dependency graph we identified the parallel sections of the program and executed them in parallel. Note that, we parallely executed the process of building the targets and not individual commands as there can be hidden dependencies between them.
To run them, one needs to pass the number of jobs using the *-j <njobs>* flag.

2 Architecture and Components

Forge is structured with clear modularity, featuring:

- **Cache**: Manages file caches, handles SQLite database interactions, and validates file integrity via SHA256.
- **Graph**: Implements a directed graph to model build target dependencies, supporting operations like cycle detection and topological sorting.
- **Parser**: Parses configuration files, resolves imports, and manages targets and variables.
- **Node and Target Structures**: Represent build targets and dependency relationships.

Build dependencies are modeled using a *directed graph*. Cycle detection is implemented via Depth-First Search (DFS), ensuring that invalid dependency structures are detected early.

3 Tools and Technologies Used

- **C++20**: Primary language, taking advantage of modern language features such as structured bindings and concepts.
- **SQLite3**: Lightweight relational database used for file caching.
- **OpenSSL**: Provides cryptographic functions for hashing.

- **Doxygen**: Used for auto-generating detailed code documentation.
- **Python 3**: Assists in HTML generation for build results.
- **Graphviz**: Used to visualize dependency graphs generated by Forge.

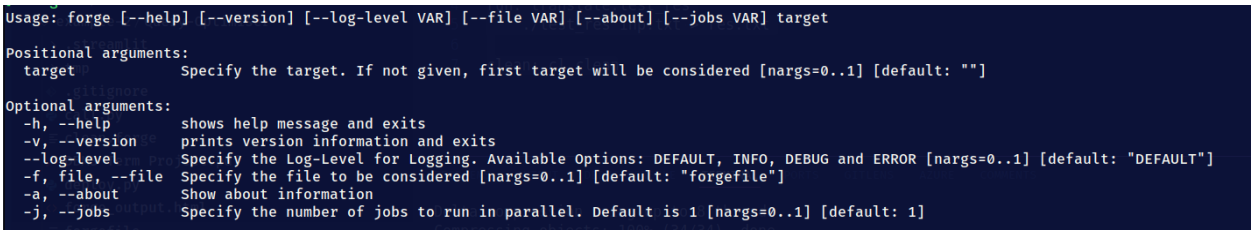
Additional libraries like *argparse* and *tabulate* streamline argument parsing and table generation.

4 Installation and Usage

Installing Forge requires a system equipped with a *C++20* compliant compiler, Python 3, SQLite3, and OpenSSL libraries. The setup follows a traditional Unix-like configuration sequence:

1. Clone the repository: *git clone https://github.com/Kronos-192081/Forge.git*
2. Install prerequisites: *./configure*
3. Build: *make*
4. Install: *sudo make install*

To use it, run *forge -h*. The output is the following:



5 Conclusion

Forge represents a modern evolution of build systems, blending modular design, high performance, and user-centric output formats. Its combination of caching, parallelism, and visual outputs make it a highly effective tool for contemporary software development projects. By leveraging technologies like SQLite, OpenSSL, and Graphviz, Forge delivers a robust, extensible, and developer-friendly build environment.
Future extensions could include more advanced dependency management, dynamic plugin systems, or further integration with continuous integration/continuous deployment (CI/CD) pipelines.