

\*\*\*

## **for educational purposes, if git were rewritten from scratch (basic features), how would you start?**

< If I were to start from scratch and rewrite Git, one approach I might take would be to begin by designing a data structure to efficiently store the file system changes that Git tracks. This data structure, often called a “content-addressable file system,” would use the contents of each file as a key to look up the file’s metadata, such as its name, file type, and timestamps.

Next, I would write the core logic for Git’s main commands: “commit,” “push,” “pull,” “diff,” etc. This would include functions to compute and apply file system deltas, handle branching and merging, and manage remote repositories.

Once the basic functionality was in place, I would focus on making the user interface as simple and intuitive as possible, while providing advanced options and hooks for power users. I would also add support for additional features such as authentication, encryption, and compression.

Finally, I would focus on performance optimization and scalability, testing the implementation with large codebases, and evaluate the performance on various operating systems and hardware configurations.

It’s worth noting that this is one possible approach and actual development of Git might be different due to the fact that there already exist multiple implementations and Git’s development history is open and public, also there is a comprehensive set of tools and libraries with well-defined interfaces and protocols that are used by Git making it a complex system to replicate from scratch.

## **great. could you list the (sorted) first 15 tasks in a jira-like style. the language is rust and the target is wasm. with these first tasks we can move from there**

< Sure, here are the first 15 tasks in a Jira-like style for a Git implementation written in Rust and targeting WebAssembly:

- Design content-addressable file system data structure: This task would involve designing a data structure in Rust to efficiently store the file system changes that Git tracks, and determining the best data structures and algorithms to use.
- Implement core logic for ‘commit’ command: This task would involve writing the core logic for the ‘commit’ command, including functions to compute and apply file system deltas and handle branching and merging.
- Implement core logic for ‘push’ and ‘pull’ commands: This task would involve writing the core logic for the ‘push’ and ‘pull’ commands, including functions to manage remote repositories and transfer data to and from them.
- Implement ‘diff’ command: This task would involve writing the logic for the ‘diff’ command, which shows the differences between commits, files, and directories.
- Implement basic user interface: This task would involve creating a basic user interface for the Git implementation, including command-line options and output formatting.
- Implement WebAssembly target: This task would involve setting up a Rust development environment targeting WebAssembly, and configuring the build process to generate the appropriate files for use in a web environment.

- 
- Implement authentication support: This task would involve adding support for authentication to the Git implementation, including functions to handle user authentication and authorization.
  - Implement encryption support: This task would involve adding support for encryption to the Git implementation, including functions to handle data encryption and decryption.
  - Implement compression support: This task would involve adding support for compression to the Git implementation, including functions to handle data compression and decompression.
  - Implement performance optimization: This task would involve optimizing the performance of the Git implementation, including profiling and benchmarking, and making changes to improve performance as necessary.
  - Test implementation with large codebases: This task would involve testing the implementation of Git with large codebases to identify and fix any scalability issues.
  - Evaluate performance on different hardware and OS: This task would involve evaluating the performance of the Git implementation on various hardware and operating system configurations to ensure it is running optimally.
  - Implement feature flagging system: This task would involve implementing a feature flagging system to toggle on/off various features of the Git implementation.
  - Implement advanced user interface: This task would involve creating an advanced user interface for power users, including advanced command-line options, output formatting and hooks.
  - Release and maintenance: This task would involve packaging, releasing and maintaining the software, providing bug fixes and updates as necessary, and working with the community for feedback and contributions.