# **Git Hooks Documentation: Automated File Metadata Management**

## **Overview**

This documentation describes a pre-commit Git hook implementation that automatically manages metadata for SQL, SQLX, and Python files. The hook tracks file modifications by maintaining a versioned history within the files themselves, using appropriate comment syntax for each file type.

## **Features**

The pre-commit hook provides the following functionality:

1. Automatic metadata insertion for new files
2. Version tracking with automatic incrementation
3. Modification history tracking
4. File-specific comment syntax
5. Commit message integration
6. Creator and modification timestamps

## **Installation**

1. Save the script as .git/hooks/pre-commit in your repository
2. Make the script executable:chmod +x .git/hooks/pre-commit

## **How It Works**

### **File Processing**

The hook automatically processes files with the following extensions:

* .py (Python files)
* .sql (SQL files)
* .sqlx (SQLX files)

### **Comment Syntax**

The hook uses appropriate comment syntax based on file type:

* Python files: Uses # for comments
* SQL/SQLX files: Uses -- for comments

### **Metadata Structure**

For a Python file, the metadata appears as:

# creator name: [Original Author]  
# creation time: [Original Timestamp]  
# Modified by Version Modified timestamp Commit message  
# --------------------------------------------------------------------------------  
# [Author] 1.0 [Timestamp] [Initial commit message]  
# [Author] 1.1 [Timestamp] [Next commit message]

For SQL/SQLX files, the format is identical but uses -- instead of #:

-- creator name: [Original Author]  
-- creation time: [Original Timestamp]  
-- Modified by Version Modified timestamp Commit message  
-- --------------------------------------------------------------------------------  
-- [Author] 1.0 [Timestamp] [Initial commit message]  
-- [Author] 1.1 [Timestamp] [Next commit message]

### **Version Control**

Version numbers follow a simple incremental pattern:

* Initial version starts at 1.0
* Minor version increments by 1 (1.0 → 1.1 → 1.2)
* When minor version reaches 10, major version increments (1.9 → 2.0)

### **Metadata Fields**

Each entry in the modification history contains:

1. Modified by: Username from Git config
2. Version: Automatically incremented version number
3. Modified timestamp: Date and time of modification
4. Commit message: Current commit message

## **Technical Implementation Details**

### **Key Functions**

1. get\_comment\_prefix():
   1. Determines appropriate comment syntax based on file extension
   2. Returns '#' for Python files and '--' for SQL/SQLX files
2. get\_next\_version():
   1. Reads the last version number from the file
   2. Calculates the next version number
   3. Handles version incrementing logic
   4. Returns the new version number
3. update\_metadata():
   1. Main function that handles metadata management
   2. Creates initial metadata for new files
   3. Updates existing metadata with new entries
   4. Maintains proper formatting and alignment

### **Error Handling**

The script includes several safeguards:

* Fallback for missing commit messages
* Validation of version number format
* Protection against invalid file content
* Proper handling of file permissions

## **Best Practices**

1. Always provide meaningful commit messages
2. Don't manually modify the metadata section
3. Ensure proper file permissions for the hook script
4. Keep the metadata section at the top of the file

## **Limitations**

1. Only processes .py, .sql, and .sqlx files
2. Requires Git user name to be configured
3. Metadata must remain at the top of the file
4. Cannot handle binary files

## **Troubleshooting**

Common issues and solutions:

1. Hook not executing:
   1. Check file permissions (should be executable)
   2. Verify hook location (.git/hooks/pre-commit)
   3. Ensure script has proper shebang line
2. Version not incrementing:
   1. Check file permissions (should be writable)
   2. Verify metadata format hasn't been manually modified
   3. Ensure proper comment syntax for file type
3. Missing commit messages:
   1. Verify Git configuration
   2. Check .git/COMMIT\_EDITMSG file exists
   3. Ensure proper Git command usage

## **Support**

For any issues or enhancements:

1. Check file permissions and Git configuration
2. Verify hook installation location
3. Ensure proper metadata format
4. Check Git user configuration

## **Future Enhancements**

Possible improvements to consider:

1. Support for additional file types
2. Configurable version numbering schemes
3. Custom metadata field support
4. Backup mechanism for metadata
5. Integration with CI/CD pipelines

## **Conclusion**

This Git hook provides automated metadata management for your repository, ensuring consistent tracking of file modifications and versions. By following the installation instructions and best practices, you can maintain a reliable history of changes in your codebase.