**Add Command: Staging Files with SHA-1**

The add operation in MiniGit is responsible for preparing files for the next commit. It mimics Git’s staging area, ensuring that only selected files are tracked and recorded.

**Steps involved in Add:**

1. **Read File Contents**: When a user runs ./minigit add <path>, MiniGit reads the file(s) located at the specified path. If the path is a directory, it recursively processes all contained files.
2. **SHA-1 Hashing**: For each file, the content is hashed using SHA-1. This hash serves as a unique identifier and filename for the blob object stored in .minigit/objects/.
3. **Staging Area**:
   * MiniGit maintains a staging\_area map, which links each file path to its blob hash.
   * It also builds a staged\_trees map that organizes files hierarchically by their directories, preparing them for inclusion in a tree object.
   * Both data structures ensure that files are staged and their relationships preserved.
4. **Persistent Storage**: The blob (file content) is written to .minigit/objects/<sha1>, enabling content-addressable storage and deduplication of unchanged files.

**Commit Command: Creating Snapshots with Metadata**

The commit operation takes all staged files and creates a new snapshot of the project’s state.

**Steps involved in Commit:**

1. **Build Tree Objects**:
   * A tree object is created from staged\_trees, representing the project directory structure.
   * Tree entries can refer to either blobs (files) or other subtrees (nested directories).
   * The root tree’s SHA-1 hash serves as the state of the project at the time of commit.
2. **Create Commit Object**:
   * The commit includes:
     + tree: the hash of the root tree.
     + parent: the hash of the previous commit (if any).
     + timestamp: the exact time of the commit.
     + message: user-supplied description of the changes.
   * This metadata is formatted and hashed to generate a unique commit ID.
   * The commit object is saved in .minigit/objects/.
3. **Update Branch Reference**:
   * The current branch (e.g., refs/master) is updated to point to this new commit.
   * The HEAD file indirectly points to the current commit via the branch ref.
4. **Clear Staging Area**:
   * After a successful commit, both staging\_area and staged\_trees are cleared.
   * The .minigit/index file is emptied to reflect that nothing is currently staged.

**HEAD Movement After Commit**

HEAD is a special pointer that always references the current commit. In MiniGit:

* HEAD points to a branch reference, like refs/master.
* That branch file stores the hash of the latest commit.
* When a new commit is made:
  + A commit object is created and stored.
  + The corresponding branch file is updated with the new commit hash.
  + Thus, HEAD indirectly moves to this new commit via the updated branch reference.

This ensures that the version history forms a chain, where each commit knows its parent, and HEAD always reflects the tip of the current branch.