Git Commit-Tree Command Implementation Explained

Overview

This file implements a JavaScript version of Git's `commit-tree` command, which creates a new commit object in Git's object database. A commit object represents a snapshot of the repository at a specific point in time, linking together a tree object (representing the file structure) with metadata about the commit.

Class Structure

CommitTreeCommand Class

```
class CommitTreeCommand {
   constructor(tree, parent, message) {
      this.treeSHA = tree      // SHA-1 hash of the tree object
      this.parentSHA = parent  // SHA-1 hash of the parent commit
      this.message = message   // Commit message
   }
}
```

The constructor takes three essential parameters:

- **tree**: The SHA-1 hash of a tree object (represents the directory structure and file contents)
- **parent**: The SHA-1 hash of the parent commit (the commit this new commit builds upon)
- **message**: A descriptive message explaining what changes this commit introduces

Core Implementation: execute() Method

1. Creating Commit Content

```
const commitContentBuffer = Buffer.concat([
   Buffer.from(`tree ${this.treeSHA}\n`),
   Buffer.from(`parent ${this.parentSHA}\n`),
   Buffer.from(`author Priyanshu Naik <priyanshunaik@Priyanshus-MacBook-Air.local>
${Date.now()} +0000\n`),
```

```
    Buffer.from(`committer Priyanshu Naik <priyanshunaik@Priyanshus-MacBook-Air.local>
${Date.now()} +0000\n\n`),
    Buffer.from(`${this.message}\n`),
]);
```

This creates the commit object content following Git's exact format:

- **tree line**: References the tree object containing the file structure
- **parent line**: References the parent commit (for commit history)
- **author line**: Author information with timestamp and timezone
- **committer line**: Committer information (often same as author)
- **blank line**: Separates metadata from commit message
- **commit message**: The actual commit message

Note: The timestamp uses `Date.now()` which returns milliseconds, but Git typically uses seconds. This might cause compatibility issues with standard Git tools.

2. Creating Git Object Format

```
const commitHeader = `commit ${commitContentBuffer.length}\0`;
const data = Buffer.concat([Buffer.from(commitHeader), commitContentBuffer]);
```

Git objects follow a specific format:

- **Header**: `<object_type> <content_length>\0`
- **Content**: The actual object data

The null byte (`\0`) separates the header from the content.

3. Generating SHA-1 Hash

```
const hash = crypto.createHash("sha1").update(data).digest("hex");
```

Git uses SHA-1 hashing to create unique identifiers for objects. The hash is computed on the complete object (header + content), ensuring data integrity and creating a unique reference.

4. Determining Storage Location

```
const folder = hash.slice(0, 2);
const file = hash.slice(2);
const completeFolderPath = path.join(process.cwd(), '.git', 'objects', folder);
```

Git uses a distributed storage system:

- First 2 characters of hash become the subdirectory name
- Remaining 38 characters become the filename
- Example: hash `abc123...` → stored in `.git/objects/ab/c123...`

This distribution prevents any single directory from becoming too large.

5. Directory Creation

```
if (!fs.existsSync(completeFolderPath)) {
    fs.mkdirSync(completeFolderPath);
}
```

Ensures the subdirectory exists before attempting to write the file.

6. Compression and Storage

```
const compressData = zlib.deflateSync(data)
fs.writeFileSync(path.join(completeFolderPath, file), compressData);
```

Git compresses all objects using zlib deflate compression to save disk space. The compressed data is written to the calculated file path.

7. Output

```
process.stdout.write(hash);
```

Returns the SHA-1 hash of the newly created commit object, which can be used to reference this commit in future operations.

Key Technical Details

Git Object Model

This implementation demonstrates Git's object model:

- **Commit objects**: Point to trees and contain metadata
- **Tree objects**: Represent directory structures
- **Blob objects**: Store file contents
- **Tag objects**: Mark specific commits

Content Addressable Storage

Git uses content-addressable storage where:

- Object content determines its hash (identifier)
- Same content always produces the same hash
- Objects are immutable once created
- References use hashes, not locations

Commit Linking

The parent SHA creates Git's commit history:

- Each commit points to its predecessor
- Creates a directed acyclic graph (DAG)
- Enables branching and merging
- Allows complete history reconstruction

Usage Context

This command would typically be called after:

1. Creating a tree object with `write-tree`
2. Having a parent commit reference
3. Wanting to record changes with a descriptive message

Potential Issues

1. **Timestamp format**: Uses milliseconds instead of seconds
2. **Hardcoded author**: Not configurable like real Git
3. **Single parent**: Doesn't handle merge commits (multiple parents)
4. **Error handling**: Limited validation of input parameters

This implementation provides a solid foundation for understanding how Git internally manages commits and demonstrates the elegance of Git's object storage system.