Git Ls-Tree Implementation

This file implements a Git command called `ls-tree` in Node.js. Let me break down what it does in detail:

Purpose

The `ls-tree` command is used to list the contents of a Git tree object, which represents a directory structure in Git's internal storage system.

Class Structure

```
class LSTreeCommand {
    constructor(flag, sha) {
        this.flag = flag;
        this.sha = sha;
    }
}
```

The class takes two parameters:

- `flag`: Command-line flags (though not actually used in the current implementation)
- `sha`: The SHA-1 hash of the Git tree object to examine

How Git Stores Objects

Git stores objects in a specific directory structure:

- Objects are stored in `.git/objects/`
- The first 2 characters of the SHA become a subdirectory name
- The remaining characters become the filename

For example, if SHA is `abc123def456`, Git stores it as `.git/objects/ab/c123def456`

Step-by-Step Execution

1. Path Construction

```
const folder = sha.slice(0, 2);     // First 2 chars: "ab"
const file = sha.slice(2);          // Rest: "c123def456"
```

```
const folderPath = path.join(process.cwd(), '.git', 'objects', folder);
const filePath = path.join(folderPath, file);
```

2. Validation

```
if(!fs.existsSync(folderPath)) throw new Error(`Not a valid object name ${sha}`);
if(!fs.existsSync(filePath)) throw new Error(`Not a valid object name ${sha}`);
```

Checks if both the directory and file exist, throwing an error if the SHA doesn't correspond to a valid Git object.

3. Reading and Decompressing

```
const fileContent = fs.readFileSync(filePath);
const outputBuffer = zlib.inflateSync(fileContent);
```

Git compresses all objects using zlib, so the file must be decompressed before reading.

4. Parsing the Tree Object

```
const output = outputBuffer.toString().split("\0");
const treeContent = output.slice(1).filter((e) => e.includes(" "));
const names = treeContent.map((e) => e.split(" ")[1]);
```

Git tree objects have a specific format:

- Start with a header like "tree 1234" (object type and size)
- Followed by null-terminated entries
- Each entry contains: mode, filename, and SHA (in binary)

The code:

1. Splits on null characters (`\0`)
2. Skips the first element (the header)
3. Filters entries that contain spaces (valid tree entries)
4. Extracts just the filenames (second part after splitting on space)

5. Output

```
names.forEach(name => process.stdout.write(`${name}\n`));
```

Prints each filename on a new line to stdout.

Example Usage

If you had a tree object with SHA `abc123...`, running this would output something like:

README.md
src/
package.json

Limitations

- The `flag` parameter is accepted but not used
- Only extracts filenames, not file modes or object types
- Doesn't handle nested tree traversal
- Error handling is basic

This is essentially a simplified version of Git's `git ls-tree` command, which is used internally by Git to examine the contents of tree objects in the repository.