MiniGit: Implementation Overview

1. Data Structures

- **`GitRepository`**
  Holds the working-tree path (`worktree`) and the `.minigit` directory (`gitdir`), plus loaded configuration. Central hub for locating files and directories in the repo.
- **Index (`GitIndex` & `GitIndexEntry`)**
  - `GitIndex`
    - `version` (int)
    - `entries` (list of `GitIndexEntry`)
  - `GitIndexEntry`
    - Metadata tuple fields (`ctime`, `mtime`, `dev`, `ino`, etc.)
    - `sha` (blob SHA1)
    - `name` (relative file path)
      Stored on disk in a binary "DIRC" format, mirroring Git's index layout.
- **Object model (`GitObject` and subclasses)**
  - **`GitObject`** (abstract)
    - `fmt` (bytes): type label
    - `serialize()`, `deserialize()`, `init()`
  - **Subclasses**
    - `GitBlob` – raw file data
    - `GitCommit` – key–value list + message (`kvlm` dict)
    - `GitTree` – directory entries (`GitTreeLeaf` list)
    - `GitTag` – annotated tag (inherits commit logic)
      Objects are stored compressed under `.minigit/objects/<prefix>/<suffix>` by SHA1.
- **`GitTreeLeaf`**
  Simple struct with `mode` (bytes), `path` (str), `sha` (hex str) for tree entries.
- **Ignore rules (`GitIgnore`)**
  - `absolute`: list of global/exclude rule-sets
  - `scoped`: map of directory → rule-set
    Patterns parsed into `(pattern, keep)` pairs via fnmatch semantics.

---

2. Key Design Decisions

1. **Modular Package Layout**
   - Top-level modules under `minigit/` for core functionality:
     - `repository.py`, `index.py`, `refs.py`, `ignore.py`, `utils.py`, plus a thin `cli.py`.

    - Two subpackages:
        - `objects/` for object-type classes & I/O
        - `commands/` for each CLI command handler
2. **Separation of Concerns**
    - **`cli.py`** handles argument parsing and dispatch to a named `cmd_<name>(args)` function.
    - **`commands/`** modules use only high-level helpers (e.g., `repo_find`, `index_read`, `object_read`) to implement behavior.
    - **`objects/`** modules focus strictly on object serialization, storage, and format parsing.
3. **Lazy imports to avoid circular dependencies**
   In `objects/base.py`, we import subclasses (`GitBlob`, `GitCommit`, etc.) inside functions rather than at module top to prevent import loops.
4. **Argparse-driven CLI**
   Subcommands mirror Git's interface (`init`, `add`, `commit`, `log`, etc.), each with its own module and clear responsibilities.
5. **Graphviz output for `log`**
   Rather than plain text, `log` emits a DOT `digraph` for easy visualization of commit history.

---

3. Limitations & Future Improvements

Limitations

- **No remote support**
  Lacks `push`/`pull` or any networking—purely local.
- **Simplified merge**
  Only three-way textual merge on blobs; no conflict markers or manual conflict resolution UI.
- **Diff limited to text**
  Binary files and large diffs aren't handled gracefully.
- **File metadata ignored**
  Permissions, symlinks, executable bits, and other metadata are not tracked.
- **Performance**
    - Linear scans of object directories for abbreviated SHAs.
    - No packfile support; storing each object as an individual file.
- **Minimal ignore integration**
  `status` doesn't consider `.gitignore` rules when showing untracked files.
- **No hooks or extensibility**
  Cannot run custom scripts on events (e.g. `pre-commit`, `post-merge`).
- **Testing coverage**
  Currently no automated tests; relies on manual smoke-testing.

Future Improvements

1. **Packfiles & indexing**
   Implement Git-style packfile storage for efficient cloning and object retrieval.
2. **Interactive conflict resolution**
   When merge conflicts occur, write conflict markers into files and provide a simple CLI prompt to resolve.
3. **Hook system**
   Allow users to define scripts in `.minigit/hooks/` for lifecycle events.
4. **Permission & symlink support**
   Track file modes, executable bits, and symbolic links.
5. **Optimized SHA lookup**
   Build an in-memory index for objects to avoid filesystem globbing when resolving abbreviations.
6. **Automated testing suite**
   Add pytest-based tests covering each command, edge cases, and file-system behaviors.
7. **`.gitignore` integration in `status`**
   Exclude ignored/untracked files from status reports.
8. **Improved CLI UX**
   - Colored output
   - Progress bars for long operations
   - Better error messages (with suggestions)
