Gitlite 项目说明

类与职责概览

- `SomeObj`（src/SomeObj.cpp）：核心命令实现类，封装 init/add/commit/rm/log/globalLog/find/checkout/status/branch/rmBranch/reset/merge 以及远程 addRemote/rmRemote/push/fetch/pull。无成员变量，所有状态通过文件系统 `.gitlite` 目录维护。
- `Utils`（include/Utils.h, src/Utils.cpp）：工具集，提供 SHA-1 计算、文件读写、目录遍历、存在性/类型检查、创建目录、错误输出与退出。主要静态常量：`UID_LENGTH = 40`（哈希长度）。无持久成员。
- `GitliteException`（include/GitliteException.h, src/GitliteException.cpp）：自定义异常，内部仅有 `std::string message` 存储错误信息，`what()` 返回 C 字符串。构造时可携带消息。
- `Repository / Commit`（头文件空）：未定义成员，功能集中在 `SomeObj`.

类的成员与静态变量概览

- `SomeObj`：无实例成员、无静态成员；所有逻辑通过静态工具和文件系统操作完成。
- `Utils`：
  - 静态常量：`UID_LENGTH = 40`
  - 静态函数：SHA-1 计算、文件/目录操作、序列化、消息/退出、存在性检测。
- `GitliteException`：
  - 实例成员：`std::string message`（存储错误信息）。

> *由于状态全部落盘到 `.gitlite`，类本身不持有内存态字段，减少了多实例时的一致性问题。*

状态与持久化设计（.gitlite 目录结构）

```
.gitlite/
```

- `HEAD`：文本，内容形如 `ref: refs/heads/master`，指向当前分支引用。
- `objects/`：存储提交与 blob。
  - blob：文件内容的 SHA-1 作为文件名，内容为原文件字节。
  - commit：提交对象，文件名为提交 SHA-1，内容文本结构：
    - `parent <p1> <p2>`（合并提交有两个父；普通提交一个父；初始提交为空字符串）
    - `timestamp <epoch_seconds>`
    - `message <msg>`
    - `files f1:blob1;f2:blob2;...;`（以分号分隔，DELETE 标记不会写入 commit；存储当前树快照）
- `refs/heads/`：本地分支引用文件，每个文件内是对应分支 head 提交的 SHA-1。
- `refs/remotes/`：远程相关引用基目录；本实现将远程跟踪分支存放在 `refs/heads/<remote>/<branch>`。
- `remotes/`：远端配置，文件名为远端名，内容为远端仓库路径字符串。

- `staging/` ：暂存区目录（若存在）。文件名为工作区路径；内容为 blob id，或字符串 `DELETE` 表示已暂存删除。

持久化示例（初始化后）

```
.gitlite/
  HEAD                      # ref: refs/heads/master
  objects/
    <init-commit-sha>       # 初始提交（空树）
  refs/
    heads/
      master                # 指向 <init-commit-sha>
    remotes/                # 预留目录，初始为空
  remotes/                  # 远端配置目录，初始为空
  staging/                  # 暂存目录，按需创建；commit/reset/merge/checkout 后被清理
```

提交对象文本格式示例

```
parent <p1> [<p2>]
timestamp 1712345678
message Merged feature into master.
files foo.txt:abc123...;bar.txt:def456...;
```

- `parent` ： 合并提交包含两个父；初始提交为空字符串。
- `files` ： 以分号分隔的 `文件名:blobId` ， 不含 `DELETE` ； 表示提交时的完整快照。

序列化/反序列化方式：

- Blob：直接写入文件（ `Utils::writeContents` ），读取用 `readContentsAsString` 。
- Commit：文本串拼装，字段行前缀固定；解析时用 `find` / `substr` 提取父、时间戳、消息、files 段并按 `name:blob;` 拆分。
- 引用/配置：纯文本（HEAD、refs/、*remotes/*）。

关键命令工作原理与边界处理

- `init` ： 创建 `.gitlite` 目录结构；生成空树的初始提交（时间戳 0，消息 “initial commit”），写入 `objects/` ，分支 `master` 指向它，HEAD 指向 master。
- `add` ： 读取工作区文件，写 blob（若不存在），若与当前提交相同则从暂存区移除；若曾暂存删除且内容相同则撤销删除；否则在暂存区记录 blob id。
- `commit` ： 要求消息非空且暂存区非空。基于当前提交的文件映射，应用暂存区（DELETE 移除，其他更新），生成新 commit 文本写入 `objects/` ，更新当前分支引用，清空暂存区。
- `rm` ： 若既未暂存也未被跟踪则报错；若仅暂存则撤销暂存；若被跟踪则在暂存区写 `DELETE` 并从工作区删除文件。
- `log` ： 沿父链打印当前分支提交（合并提交打印两个父的短哈希）。
- `globalLog` ： 遍历 `objects/` ，过滤出包含 `parent` 前缀的文件视为提交，逐个打印。
- `find` ： 遍历所有提交，匹配 message 输出提交 id，未找到时报错。
- `checkoutFile` / `checkoutFileInCommit` ： 解析（可短哈希）找到提交，提取文件对应 blob 覆盖工作区，若不存在则报错。
- `checkoutBranch` ： 切换分支前检查是否有未跟踪文件会被覆盖；将目标提交的所有文件写入工作区，并删除当前提交有而目标没有的文件；更新 HEAD；清理暂存区。

- `status`：
  - 分支：列出并标记当前分支。
  - 暂存：列出 staging 内非 DELETE；删除：列出 staging 内 DELETE。
  - 未暂存修改：对工作区、tracked、staged 三方比对，找出内容变化或缺失但未标记 DELETE 的文件。
  - 未跟踪：工作区中既未暂存也未跟踪的文件。
- `branch` / `rmBranch`：创建/删除分支引用（禁止删除当前分支）。
- `reset`：解析短哈希，检查提交存在；保护未跟踪文件不被覆盖；将目标提交文件写入工作区，删除多余文件，更新分支引用并清空暂存区。
- `merge`：
  - 前置：仓库已初始化、目标分支存在、不同于当前分支、暂存区必须为空。
  - 找 split point；若给定分支是祖先则提示退出；若当前分支是祖先则快进到给定分支。
  - 三方合并：遍历 split/current/given 的所有文件集合，按修改性（相对 split）决策：
    - 仅给定修改：用给定版本写工作区并暂存。
    - 仅当前修改：保留当前。
    - 同改同内容：无操作。
    - 删除场景按 split/当前/给定组合处理（保持删除或报冲突）。
    - 冲突：写入带 `<<<<<<< HEAD` / `=======` / `>>>>>>>` 分隔的内容，生成 blob、写工作区并暂存。
  - 若有冲突打印提示；若最终暂存为空则报错；创建合并提交（两个父），更新当前分支，清理暂存区。
- 远程：
  - `addRemote` / `rmRemote`：在 `.gitlite/remotes` 下记录/删除远端路径。
  - `push`：读取远端路径，要求远端分支 head 是本地 head 的祖先（快进要求），否则提示先拉取。BFS 复制本地提交与关联 blob 至远端 objects，再更新远端分支引用。
  - `fetch`：BFS 从远端分支 head 复制提交与 blob 到本地 objects，不改工作区，更新本地跟踪引用 `refs/heads/<remote>/<branch>`。
  - `pull`：先 fetch，再 merge 远端跟踪分支到当前分支，复用本地 merge 冲突处理。

三方合并决策表（相对 split）

split	current	given	结果
same	same	changed	取 given，写工作区并暂存
same	changed	same	保留 current
same	changed	changed(同)	保留（无操作）
same	changed	changed(不同)	冲突，写冲突标记并暂存
present	deleted	same	保持删除（若 current 未改）
present	same	deleted	保持删除（若 given 未改）
absent	present	present(同)	取任一（无冲突）

split	current	given	结果
absent	present	present(不同)	冲突

> *“modified” 判定基于 blobId 是否与 split 不同；删除视为不在文件映射中。*

远程同步算法要点

- `push`
  1. 读取远端路径；远端分支若存在，必须是本地 head 的祖先（快进）。
  2. 自本地 head 做 BFS，将提交与引用的 blob 写入远端 `objects/` （缺啥补啥）。
  3. 更新远端 `refs/heads/<branch>` 指向本地 head。
- `fetch`
  1. 读取远端路径与分支，获取远端 head。
  2. 从远端 head BFS 复制提交与 blob 到本地 `objects/` ，不触碰工作区。
  3. 更新本地跟踪引用 `refs/heads/<remote>/<branch>` 。
- `pull` 先 fetch，再 merge 跟踪分支到当前分支，冲突处理与本地 merge 相同。

暂存区语义

- 文件内容变更：暂存条目存 blobId。
- 删除：暂存条目写 `DELETE` 。
- `commit` 会应用暂存条目到当前快照并清空暂存； `reset/checkoutBranch/merge` 结束后也清理暂存以保证工作区与引用一致。

短哈希解析

- 在 `checkoutFileInCommit` 、 `reset` 等场景，若传入 ID 长度 < 40，则遍历 `objects/` 找到前缀匹配的唯一提交；未找到或歧义则报错。

测试驱动（testing/tester.py 指令语法）

> *测试器会读取 `*.in` 脚本，按指令驱动 `gitlite` 可执行文件，并比对输出/文件。*

- `#` ：注释。
- `I FILE` ：包含 FILE 的内容（相对当前 .in 的目录）。
- `C DIR` ：切换到子目录 DIR；空值切回根目录（用于模拟远端）。
- `T N` ：将后续命令超时设为 N 秒。
- `+ NAME F` ：拷贝 `src/F` 为工作区文件 NAME。
- `- NAME` ：删除工作区文件 NAME。
- `> COMMAND ARGS ...` + 输出 + `<<<[*]` ：执行 `gitlite COMMAND ...` ，比较输出；`<<<*` 表示期望为正则；否则用编辑距离容差（默认 0，可用 `--tolerance` 调整）。
- `= NAME F` ：断言工作区文件 NAME 等同于 `src/F` 。
- `* NAME` ：断言文件 NAME 不存在。

- `E NAME`：断言文件或目录 NAME 存在。
- `D VAR "VALUE"`：定义变量，可在脚本中用 `${VAR}` 或 `${N}`（最近一次正则匹配分组）替换。

运行要点：

- 默认可执行为 `build/gitlite`；可用 `--progdir` 指定目录。
- `--show` 控制失败详情；`--keep` 保留测试生成的临时目录；`--src` 切换样例基目录；`--reps` 重复测试；`--debug` 逐条命令交互式执行。

边界与错误处理摘要

- 缺文件/目录或未初始化：调用 `Utils::exitWithMessage` 终止并输出原因。
- 短哈希解析：在 `checkoutFileInCommit`、`reset` 等处遍历 `objects/` 以补全。
- 未跟踪文件保护：`checkoutBranch`、`reset`、`merge` 中若有未跟踪文件会被覆盖则直接退出提示。
- 暂存必须为空：`merge` 前置校验；`commit` 需暂存非空。

持久化文件计数（初始化后最小集）

- 根：`.gitlite/HEAD`（1）
- 引用：`.gitlite/refs/heads/master`（1，更多分支则增加）
- 对象：`.gitlite/objects/<initial-commit-id>`（1 提交）；后续提交/文件会增加 blob/commit 文件。
- 配置：`.gitlite/remotes/`（0+，按远端数量）；远端跟踪引用存放于 `.gitlite/refs/heads/<remote>/<branch>`，按 fetch 的分支数增加。
- 暂存：`.gitlite/staging/` 按需存在并包含已暂存的文件条目；在 commit/reset/checkout/merge 等操作后被清理。

如需进一步细化（例如逐命令的输入输出示例、错误提示清单），请告知。