基本操作

由于小蓝鲨水平不够，git-- 只支持对文件的操作，并不支持目录（文件夹），同时 git—支持的文件操作只有三个：写入、读取和删除。这三个操作均依赖于文件的查找。

文件的查找　查找一个文件 X 的流程如图所示，具体规则如下：1. 如果暂存区中能够查找到 X 文件，则找到了该文件；2. 如果暂存区中不存在 X 文件，但是存在其暗文件 .X，则表示目标文件被删除，返回文件已被删除；3. 如果在暂存区中，既不存在 X 文件，也不存在其暗文件，则在暂存区中无法确定是否存在该文件，需继续查找头部所指向的提交。如果头部为空，则文件不存在；4. 在一个提交中查找文件的方法与在暂存区中查找的方法相同，即先后检查目标文件和其暗文件。如果该提交中依然无法确定是否存在该文件，则继续查找该提交的父提交，直至能够确定目标文件的存在性，或父提交不存在，则该文件不存在。对于具有两个父提交的提交，其查找方法在后文 merge 命令中给出。

写入命令　共包括两行输入，其中第一行格式为 write <filename> <offset> <len>（其中尖括号表示参数，下同），表示向文件 <filename> 中写入数据，写入的数据长度为 <len> 个字节，写入的开始位置为文件的 <offset> 位置（注意，在文件的位置 0 写入 1 个字符，写入的是文件的第 1 个字符）。如果 <offset> 的位置超出了该文件的大小，则从文件当前末尾到 <offset> 之间的区域使用 ASCII 字符点（.）进行填充。

在此行命令之后的一行，用户会输入 <len> 个字节作为文件内容，（注意结尾会有一个换行符 \n，不算做文件内容）。用户输入的内容中不包含换行字符，但是**可能包含空格**。

在执行写入命令时，git-- 首先按照前述方法查找该文件，并根据不同情况进行不同操作：

- 如果在暂存区中该文件被找到，则直接进行写入操作；
- 如果在某个提交中找到了该文件，则先将找到的文件拷贝到暂存区中（即在暂存区中创建该文件，并将找到的文件的内容拷贝到新文件中），再在暂存区中的文件中进行写入操作；
- 如果查找结果为文件被删除，如果是在暂存区中被删除，则删除暂存区中对应的暗文件，并在暂存区中创建该文件后进行写入操作；
- 如果是在某个提交中被删除，则在暂存区中创建该文件后进行写入操作；
- 如果该文件不存在，则在暂存区中创建该文件，并进行写入操作。

写入命令不产生输出。

读取命令　共占一行，格式为 read <filename> <offset> <len>，表示从文件 <filename> 的 <offset> 位置开始读取此后的 <len> 个字节。对于超出文件当前大小的部分，每个超出的字节以一个 ASCII 字符点（.）替代。在执行时，git--首先按前述方法查找该文件，如果能找到文件，则输出文件中对应的内容；如果文件不存在或者文件被删除，则输出 <len> 个 ASCII 字符点（.）。文件读取命令的输出共占一行，因此在文件内容后应有换行（\n）字符，格式也可以参考样例。

删除命令　占一行，格式为 unlink <filename>，表示删除名为 <filename> 的文件。如果 git-- 中无法找到该文件或该文件被删除，则什么都不做。如果能够找到该文件，则在暂存区中添加该文件的暗文件。如果目标文件是在暂存区中被找到的，则还需要从暂存区中删除目标文件，只保留其暗文件。

删除命令不产生输出。

注意，删除操作并不能抵消文件创建操作的效果。在文件 X 不存在的情况下，用户可以首先创建文件 X，之后将文件 X 删除。在删除后，暂存区中会存有一个 X 的暗文件（.X），与文件 X 被创建前的状态不同。

列举命令　　占一行，格式为 ls，输出在当前的暂存区和当前的头部状态下，用户能够读到的文件（即可以查找到的文件，不包括暗文件）个数，以及其中按字典序排列，名字最小的文件名和名字最大的文件名。文件个数和两个文件名之间以一个空格隔开，共占一行。如果用户能读到的文件数为 0，则只需输出数字 0，占一行，无需给出文件名。列举命令的输出共占一行，因此在列举内容之后应有换行（\n）字符，具体可以参考样例。

高级操作

由于小蓝鲨致力于模仿 git 系统，git—系统不仅仅支持上述基本命令，还支持一系列与 git 命令相似的高级命令。

提交命令（commit）　　占一行，格式为 commit <cmtname>。
commit 命令将暂存区中的修改进行提交，其接受一个字符串类型参数 <cmtname>，为新提交的名称。在进行提交时，git--将当前的暂存区 uncommitted 重命名为给定的提交名称，并更新元数据信息。新的提交（<cmtname>）的父提交为此时头部所指向的提交。如果此时头部为空，则新提交没有父提交。此后，git-- 将更新头部，让其指向刚刚创建的新提交（<cmtname>）。最后，git-- 还会创建一个新的空暂存区，用于保存此后的修改。

注意，如果在提交时暂存区为空，或名为 <cmtname> 的提交已经存在，则该命令执行失败，git—中不产生任何修改。提交命令不产生任何输出。

切换命令（checkout）　　占一行，格式为 checkout <cmtname>。
checkout 接受一个参数，为提交名 <cmtname>。该命令将当前的头部指向 <cmtname>。在支持该命令之后，提交之间的关系可能会“分叉”。checkout 命令不一定会成功。若在进行 checkout 时，暂存区不为空，或者名为 <cmtname> 的提交不存在，则 checkout 命令执行失败，git-- 中不应产生任何修改。

合并命令（merge）　　占一行，格式为 merge <mergee> <cmtname>。

merge 命令接受两个参数，分别为需要合并的提交名 <mergee> 和新提交的名<cmtname>。假设此时头部指向的提交为 headcmt，该命令将 <mergee> 中的内容合并到提交 headcmt 之上。具体来说，GeetFS 会创建一个新的提交，名为 <cmtname>，其两个父提交为 headcmt 和 mergee。由 merge 命令创建的提交中不包含任何文件和数据，只记录了两个父提交，表示这两个父提交的内容在逻辑上进行了合并。

注意，如果在进行 merge 时满足以下任何一个条件，则 merge 执行失败，git—中不产生任何修改:
- 失败条件 1：暂存区不为空；
- 失败条件 2：<mergee> 与 headcmt 为同一个提交；
- 失败条件 3：名为 <mergee> 的提交不存在。

支持 merge 命令会影响文件查找的规则：在支持 merge 命令后，一个提交（cmt）可以有两个不同的父提交（cmt.parent1 和 cmt.parent2）。在进行文件查找时，若在 cmt

中无法确定目标文件是否存在，则 git-- 需要通过 cmt.parent1 和 cmt.parent2 两个父提交分别进行文件查找:

- 若通过两个父提交均无法找到目标文件或其暗文件，则表示要查找的文件不存在；
- 若仅能通过其中一个父提交找到该目标文件或其暗文件，则以此找到的文件作为文件查找的结果；
- 若通过两个父提交均能找到该目标文件或其暗文件，则根据所找到的两个文件的所在提交的创建时间进行选择，取创建时间最近（最大）的文件作为文件查找的结果。如果所找到的两个文件为同一个文件（即在同一个提交中），则以此文件作为文件查找的结果。