自从看了欧阳助教的 Intro-to-git 视频，小蓝鲨陷入了对 git 的沉迷无法自拔。恰好，最近小蓝鲨学习了一些文件系统的一些知识，于是小蓝鲨决定在寒假实现一个类似于 git 的文件系统:"git--"。由于小蓝鲨水平不够，git--仅需要完成 git 的部分命令。

git-- 是一个基于内存的文件系统，其使用内存中的结构来保存和管理用户的文件数据。下图展示了 git-- 的工作流程和 git-- 保存数据的基本方式。在使用 git-- 时，用户发送请求给 git--。git-- 会解析用户的请求，在内存数据结构中进行操作，完成用户请求。

git--中的一些基本概念：

• 文件：与普通文件系统中的文件概念相同，git-- 中的文件可以保存数据。git-- 中的文件支持读取、写入、删除。写入一个不存在（或此前已被删除）的文件会自动创建该文件

• 文件名：每个文件拥有一个文件名，为一个长度不超过 128 的字符串。文件名的内容仅包含大小写英文字母（A~Z 和 a~z）和数字（0~9）。小蓝鲨认为 git-- 的用户都非常善良，所以他们能够保证每个 git-- 命令中的文件名均满足上述规定，git-- 在实现的时候无需进行检查。

• 暗文件：为了表示文件的删除，git-- 中引入了暗文件的概念。对于一个名为 filename 的文件，其暗文件名称为 - filename（即在文件名前增加了一个负号 -）。由于在用户创建的文件名中不允许出现 -，暗文件的文件名与普通文件的文件名并不会混淆。暗文件的文件大小为 0，其中不可保存数据。其仅表示名为 filename 的文件被删除。一个文件与其暗文件不应同时出现在同一个提交中，也不应同时出现在暂存区中（关于提交和暂存区的概念请看下面两条）。如下图中的 - file2 为一个暗文件，表示对文件 file2 的删除。

• 暂存区：用户所有的修改，包括文件写入、删除等，在未提交时，均保存在暂存区（即下图中的 uncommitted 结构，包括虚线椭圆及其右侧的文件）。其中文件的写入（包括创建）以文件的形式保存，而文件的删除以暗文件的形式保存。

• 提交：与在 Git 中类似，一个提交表示 git-- 的一个历史状态。一般来说，提交由 commit 命令创建，git-- 将当前暂存区中的所有修改保存下来，并赋予一个唯一的提交名，成为一个提交。提交名的命名要求与文件名相同，且无需进行格式检查。除了 commit 命令外，用户还可以通过 merge 命令创建一个提交。通过 merge 命令创建的提交将两个现有提交进行合并。提交的创建在后文中有具体描述。除了 git-- 中的第一个提交不存在父提交之外，其余每个提交拥有一个父提交（由 commit 命令创建）或者两个父提交（由 merge 命令创建）。如下图中的 cmt1 表示一个名为 cmt1 的提交。

• HEAD：与 Git 中类似，HEAD 表示当前的头部，指向头部提交。用户当前能够访问哪些文件，以及文件的内容，取决于当前缓存区中的内容以及当前头部所指向的提交。

• git-- 命令：用户使用 git-- 命令对 git-- 的内容进行操作。命令只能通过标准输入传递给 git--，除了 write 命令占据两行之外，其他 git-- 均只占一行（即以换行符结尾）。

git-- 的存储结构

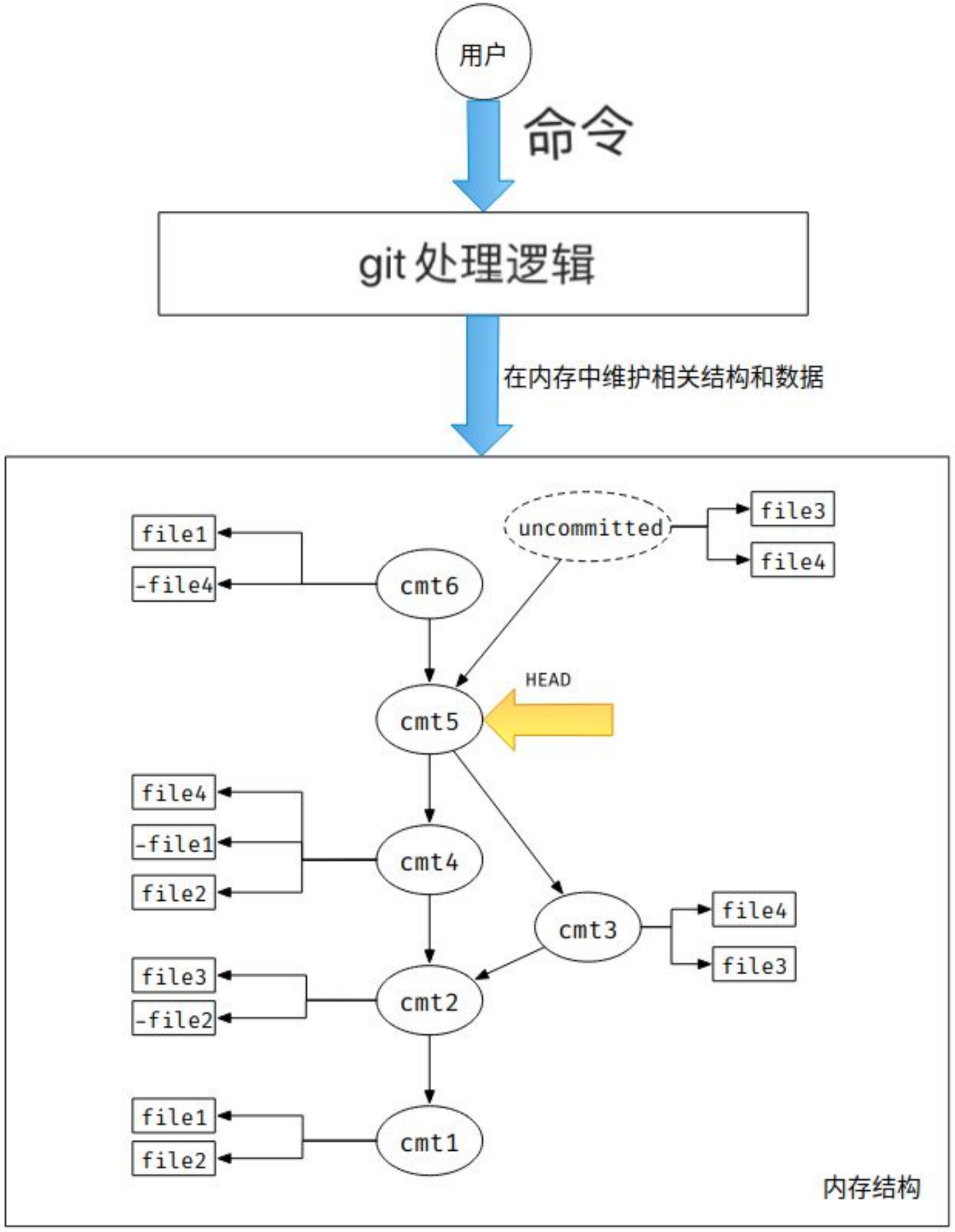
git—中保存了一个头部，一个暂存区结构，和若干个提交结构。

• **头部（HEAD）**：git-- 中有且仅有一个头部，为指向当前头部提交的一个指针或者引用，当 git—中不存在任何提交时，头部为空。

• **暂存区结构（uncommitted）**：git--中有且仅有一个暂存区结构。暂存区结构中包含了所有还未提交的修改。在此结构中，应该保存所有被修改（包括创建）文件的名称、大小和内容。对于删除的文件，该结构中应当保存对应的暗文件的信息。注意暗文件的大小为 0，不保存数据，因此只有暗文件的文件名是有意义的信息。

• **提交结构**：git-- 中可以有零个或者多个提交结构，保存在元数据结构之中。暂存区结构中的内容在提交后，变为提交结构。一旦生成，提交结构中的内容是不可修改的。

初始状态 一个刚刚被创建出来的 git--文件系统只包括一个空头部和一个空的暂存区结构（即其中没有任何文件或暗文件）。


用户
命令
git 处理逻辑
在内存中维护相关结构和数据
file1
-file4
cmt6
uncommitted
file3
file4
HEAD
cmt5
file4
-file1
file2
cmt4
cmt3
file4
file3
file3
-file2
cmt2
file1
file2
cmt1
内存结构

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>。

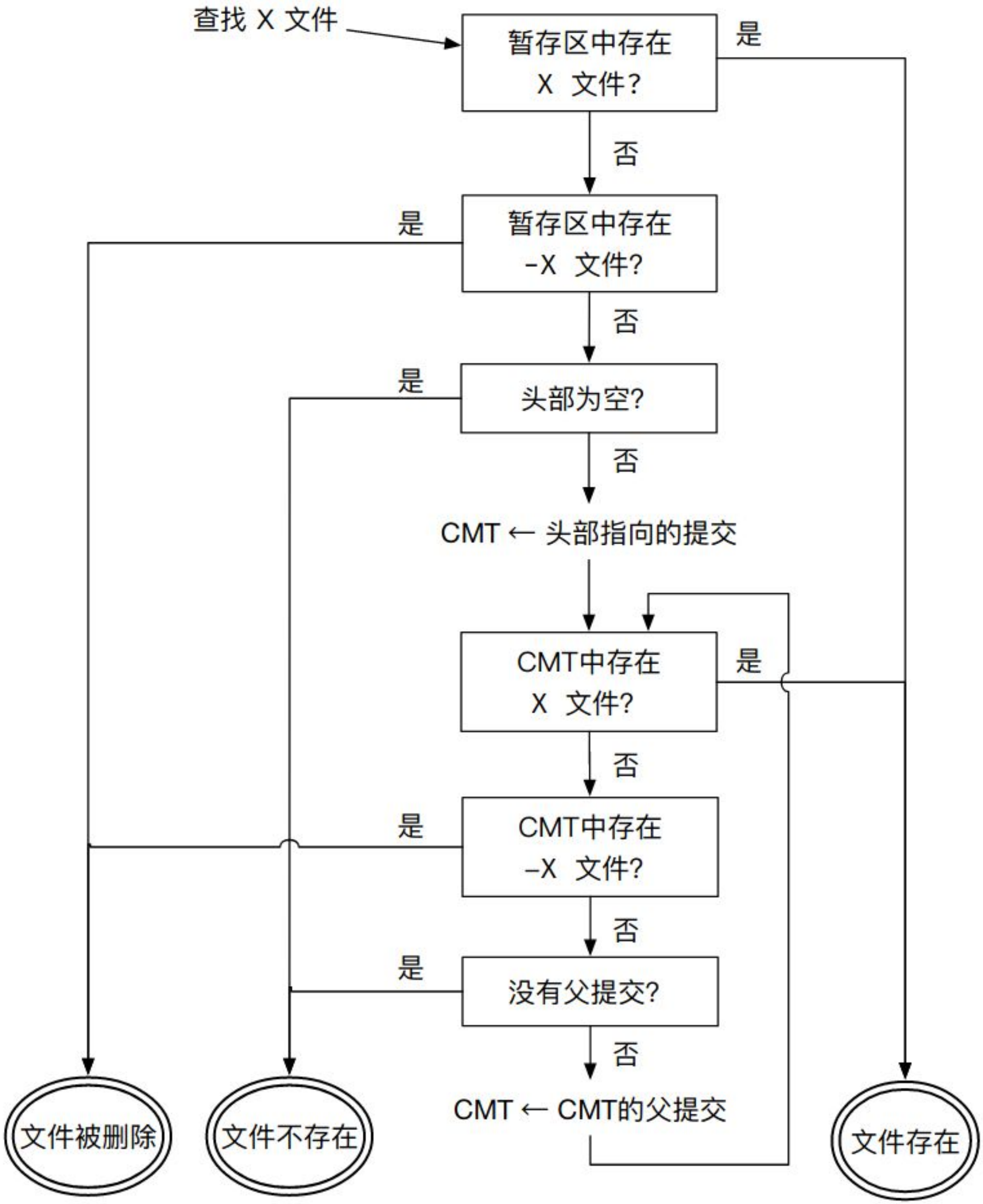
merge 命令接受两个参数，分别为需要合并的提交名 <mergee> 和新提交的名<cmtname>。假设此时头部指向的提交为 headcmt，该命令将 <mergee> 中的内容合并到提交 headcmt 之上。具体来说，GeetFS 会创建一个新的提交，名为 <cmtname>，其两个父提交为 headcmt 和 mergee。由 merge 命令创建的提交中不包含任何文件和数据，只记录了两个父提交，表示这两个父提交的内容在逻辑上进行了合并。

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。


查找 X 文件
暂存区中存在
X 文件?
是
否
暂存区中存在
-X 文件?
是
否
头部为空?
是
否
CMT ← 头部指向的提交
CMT中存在
X 文件?
是
否
CMT中存在
-X 文件?
是
否
没有父提交?
是
否
CMT ← CMT的父提交
文件被删除
文件不存在
文件存在

查找文件的流程

【输入格式】

从标准输入读入数据。

输入的第一行为一个数字 N，表示该测试中的命令总个数。从第二行开始，标准输入共包含 N 个命令。注意，每个写入命令 write 共占两行，其中第一行为写入命令以及其参数，第二行为需要写入的文件内容。文件内容中不会包含换行字符，但是可能会包含空格。其他每个命令占一行，具体格式在前文已经给出。命令均符合相应格式，文件名和提交名均符合规范，读写命令中的长度均大于 0，因此无需对命令格式进行错误处理。

【输出格式】

输出到标准输出。

根据每个命令的要求进行相应输出。

【样例 1 输入】

```
**********
10
write file1 5 2
78
write file2 7 4
abcd
read file1 0 10
ls
read file2 4 10
unlink file2
ls
read file2 3 4
write file2 1 2
12
read file2 0 4
**********
```

【样例 1 输出】

```
**********
.....78...
2 file1 file2
...abcd...
1 file1 file1
....
.12.
**********
```

样例 1 为简单的文件读写测试。

后续样例会逐步放入 OJ 网站。

子任务

子任务，本题目中，共包含多个测试点，每个测试点包含的命令类型、测试规模如下：

测试点	包含命令类型	测试规模
1	read+write	超微量
2	read+write	微量
3	read+write	中量
4	write+ls	超微量
5	write+unlink+ls	超微量
6	read+write+unlink+ls	微量
7	read+write+unlink+ls	中量
8	read+write+unlink+ls+commit+checkout	微量
9	read+write+unlink+ls+commit+checkout	中量
10	read+write+unlink+ls+commit+checkout+merge	微量
11	read+write+unlink+ls+commit+checkout+merge	中量
12	read+write+unlink+ls+commit+checkout+merge	中量
13	read+write+unlink+ls+commit+checkout+merge	海量
14	read+write+unlink+ls+commit+checkout+merge	海量

测试规模说明

测试规模	超微量	微量	中量	海量
最大文件大小	1KiB	1KiB	256KiB	2MiB
总文件数量	10	1000	5000	5000
总命令数量	10	1000	10000	20000
读写命令中的长度	100	100	100	100
总提交个数	10	100	2000	5000
所有文件的总大小	1KiB	100MiB	1GiB	1.5GiB

表格中**总文件数量**为 write 命令中所出现过的不同文件名的个数。表格中**所有文件的总大小**为整个系统中所有文件的大小之和。

每个测试点时限：3000ms
内存限制：2GiB

温馨提示：建议先实现基础命令，再逐步实现高级命令