项目要求

- 命令行工具
- 参考git实现原理，实现blob,tree,commit核心存储结构

功能点：

1. 可以提交commit，可以进行“git log”查看commit历史
2. 可以进行“git reset”回滚到指定commit
3. 可创建多分支，可在分支之间切换
4. 可考虑merge功能、远程仓库功能

tasks

task1：实现key-value存储

- 最简单的key-value存储方式
  - key作为文件名，文件内容作为value
- 支持以下功能
  - 给定value，向存储中添加对应的key-value
  - 给定key，查找得到对应的value值
- 封装成class对外提供接口
- 单元测试

task2：将一个文件夹转化为key，value

- 给定一个文件夹目录，将其转化成若干tree和blob
  - 深度优先遍历此目录
    - 遇到子文件就转化为blob并保存
    - 遇到子文件夹就递归调用文件夹内部的子文件/文件夹最后构造tree并保存
- 使用步骤1提供的接口
- 单元测试

task3：实现Commit

- 给定一个工作区目录，生成对应的blob和tree以及commit
- 编写设计文档
- 提示：
  - 需要存储指向当前最新commit的HEAD指针
  - 每次新生成一个commit前，需要把根目录的tree key与已有最新commit的tree key进行比较，发现不同时（即文件放生了变动）才添加这个commit

Git实现原理

- Key-value存储
  - value：Object的内容
  - Key：Object内容的hash
- Object的三种类型
  1. Blob：文件
     - Blob的Value：文件内容（不包含文件名）
     - Blob的Key：文件内容的hash值
  2. Tree：文件夹
     - Tree的Value：1.子文件夹和子文件名称；2.每个子文件Blob key；3.每个子文件夹tree的key；
     - Tree的Key：Tree的Value的hash值
  3. Commit：提交
     - Commit的Value：仓库tree对象的key、前驱commit对象的key、时间戳、备注
     - Commit的Key：Commit的Value的hash值

idea

1. Hash

求hash值的类

- 计算blob的hash值
  - setHashValue(File inputFile)
  - 参数为File类型
- 计算字符串的hash值
  - setHashValue(StringBuilder value)
  - 参数为StringBuilder类型
- 将得到的hash值从消息队列中取出
  - getHashValue(MessageDigest complete)
  - 返回值为16进制字符串hash值

2. KeyValue父类

实现key-value存储

- 计算出key值
  - getKey()
  - 调用Hash类里的getHashValue()方法

    - 返回值为16进制字符串hash值
- 创建key-value文件
    - 文件名为key值
    - 创建tree模块的Key-Value文件，createKeyValue(StringBuilder)
    - 创建Blob模块的Key-Value文件，createKeyValue()
        - 使用缓冲区写入Blob文件
- 功能实现
    - 判断给定的key是否对应一个有效的key-value文件
        - isValuableKey()
    - 给定key，查找得到对应的value值
        - getValue()
- 获取commit中根文件夹的hash值
    - 存储指向当前最新commit的HEAD指针
        - createHead()
    - 从Head文件中获取最新一次commit的key值
        - getLatestCommit()
        - 返回值为最新一次commit的key值
    - 获取根文件夹的hash值
        - getLatestKeyOfHomeFolder()
        - 参数为最新一次commit的key值
        - 返回值为根文件夹的hash值

3. Blob类

继承KeyValue父类

- 创建blob类型的key-value文件
    - 调用父类的createKeyValue()方法

4. Tree类

继承KeyValue父类

- 创建Tree类型的key-value文件
- 深度优先遍历文件夹，不断更新Tree的value，生成key
    - updateValue()
    - 排除".versionManagement"下的文件
    - updateEntry(File f)
- 调用父类createKeyValue(StringBuilder)方法创建文件夹的key-value文件

5. commit 类

判断commit是否进行了更新

commit

- 判断commit是否有变化
  - isValidCommit()
  - 调用getLatestKeyOfHomeFolder()方法获取根文件夹的hash值
  - 通过Tree类获取当前仓库的key值
  - 将根文件夹的hash值与当前仓库的key值进行比较
- 若commit值有变化，进行更新
  - updateValue() 更新commit的value值
  - createKeyValue()　创建最新commit的key-value文件
  - createHead();　更新（创建）HEAD文件，存储这次commit的Key

6. version managemen

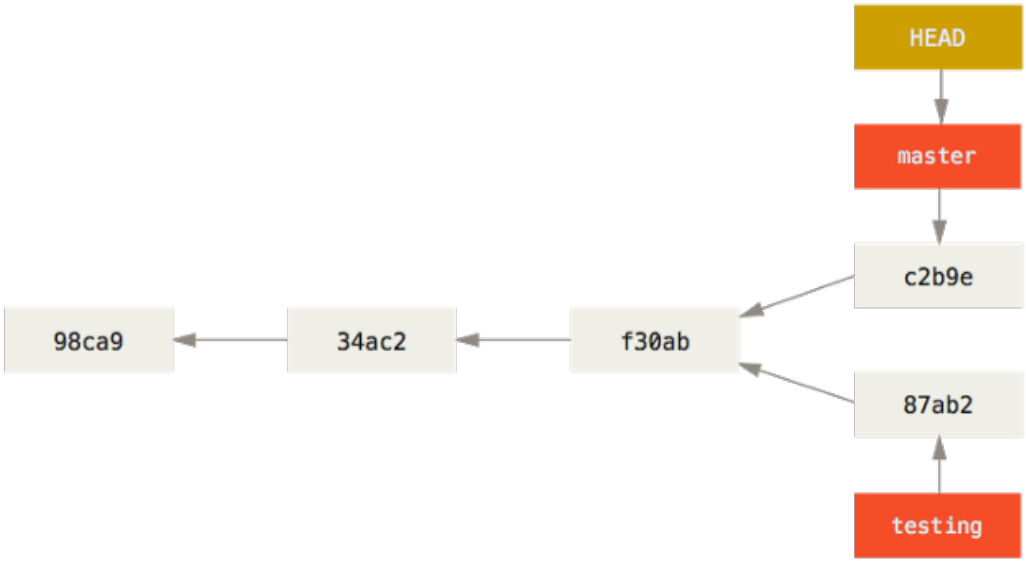
接入对仓库的操作

- 建立仓库
  - setWarehouse(String warehousePath)
  - 建立仓库下".versionManagement"文件夹，用于放置创建的key-value文件

分支管理与切换

- branch文件夹

  用于查看、修改分支信息，其中保存：
  - 所有的分支
  - 每个分支最新的commit值
  - 当前处于哪个分支
- 实现步骤
  - 把原本的HEAD指针改为branch，名称为分支的名称
  - 所有记录branch的文件都存在了branch文件夹里，名称为分支名，内容记录最新的一次commit。
  - 然后再用一个HEAD指针指向当前的分支，切换分支时便修改HEAD的值



回滚

- 思路：

  首先链表遍历出所有的commitID，然后选择回滚至其中一个commit，根据这个commit的根目录hash值可以遍历出所有的文件，然后恢复全部的文件

- 具体实现：

  把commit对应的根目录Tree对象恢复成一个文件夹

  - 根据commit key查询得到commit的value
  - 从commit value中解析得到根目录tree的key
  - 恢复(path)：
    - 根据tree的key查询得到value
    - 解析value中的每一条记录，即这个tree对象所代表的文件夹内的子文件与子文件夹名称以及对应的blob/tree key
    - 对于blob，在path中创建文件，命名为相应的文件名，写入blob的value
    - 对于tree，在path中创建文件夹，命名为相应的文件夹名，递归调用恢复(path+文件夹名)
  - 更新HEAD指针
    - 将HEAD文件中的commit的key改为回滚到的commit的key

命令行交互

- 具体实现
  - 通过main函数命令行参数String[] args接收用户指令
- 编译

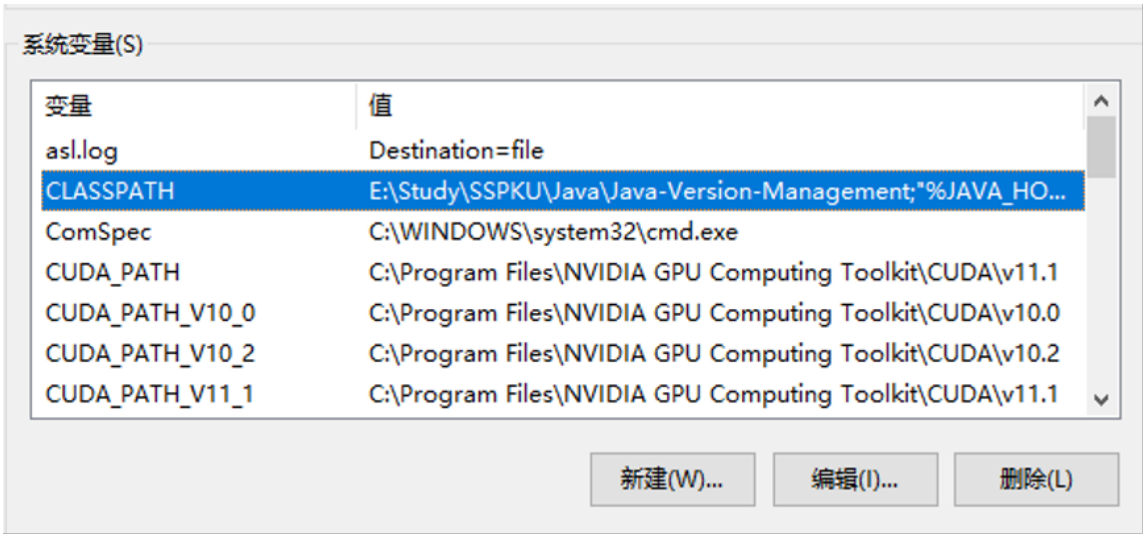
  首先在程序文件夹下编译所有的程序：

  `javac -encoding UTF-8 *.java`
- 设置环境变量
  - 在系统变量中添加

    变量名：CLASSPATH

    变量值：.class文件所在的路径

    保证.class在任意目录下都能运行



- 程序运行
  - 建立仓库（初始化）

    切换到需要建立仓库的文件夹，运行下面的语句建立仓库（初始化）

    `java gitCommand init`

    .
  - commit 功能

    `java gitCommand commit`

```
C:\Users\zrc5\Desktop\test>java gitCommand commit
请输入备注:
first commit
更新成功!
Commit 完成
```

.

- 查看分支

  `java gitCommand branch`

```
C:\Users\zrc5\Desktop\test>java gitCommand branch
目前已有分支:
Main
当前所在分支: Main
```

.

- 创建新分支

  `java gitCommand branch 分支名`

```
C:\Users\zrc5\Desktop\test>java gitCommand branch new
创建分支new成功!
```

.

- 切换分支

  `java gitCommand checkout 分支名`

```
C:\Users\zrc5\Desktop\test>java gitCommand checkout ne
切换分支失败

C:\Users\zrc5\Desktop\test>java gitCommand checkout new
切换分支成功
```

.

- 查看commit记录

  `java gitCommand log`

```
C:\Users\zrc5\Desktop\test>java gitCommand log
Commit ID: [10a32eab8230140ea77e14aa902f36cbad94e5ee, 7152810be9201111967e58b3fbedf26d604dbbd3]
```

.

- 回滚功能

  `java gitCommand rollback`

```
C:\Users\zrc5\Desktop\test>java gitCommand rollback
Commit ID: [1e4a48c3d2bf6785f4b0bdc513351a8cca98a2c6, 7152810be9201111967e58b3fbedf26d604dbbd3]
输入Commit ID:
7152810be9201111967e58b3fbedf26d604dbbd3
回滚完成!
```

.