Simple Jit

一、背景与成果

- 背景：Simple Jit是基于Java的版本控制工具
- 技术定位：初级
- 目标群体：开发人员
- 技术应用场景：在本地 .jit 文件夹内维护历史文件
- 项目设计与亮点：设计参考git的原理，主要亮点在于add操作仅hash并序列化blob对象，这样工作区中的文件在版本库中只存储一遍，保证了占用空间不会随文件夹数量指数级增长

二、操作步骤

2.1 开发前的准备工作

> - 熟悉助教提供的基本框架
> - 运行git的各种命令，观察本地库中文件的变化，确定git的原理

2.2 进入开发阶段

1. Jit init 命令

- **代码逻辑**：在工作区下创建一个名为.jit的版本库
- **参数列表**：

参数	含义
repoPath	工作区路径

Java

```java
public void createRepo() throws IOException {
    FileCreation.createDirectory(getWorkTree(),".jit");//创建仓库目录
    FileCreation.createFile(getGitDir(),"COMMIT_EDITMSG", null);//创建保存commit信息的文件
    FileCreation.createFile(getGitDir(),"config",null);//创建配置文件
    FileCreation.createFile(getGitDir(),"HEAD","ref: refs/heads/main");//创建“HEAD”文件，存储当前HEAD指向的分支
    FileCreation.createFile(getGitDir(),"description",null);//创建仓库描述文件，存储仓库名称等信息
    FileCreation.createDirectory(getGitDir(),"objects","info");//创建objects目录。保存blob,tree,commit的hash文件
    FileCreation.createDirectory(getGitDir(),"objects","pack");
    FileCreation.createDirectory(getGitDir(),"refs","heads");//创建refs目录，存储各分支信息
    FileCreation.createDirectory(getGitDir(),"refs","tags");
    FileCreation.createDirectory(getGitDir(),"logs");//创建logs目录，存储不同分支下的commit记录
    FileCreation.createDirectory(getGitDir(),"info");//创建info目录，存储仓库的其他信息
    FileCreation.createDirectory(getGitDir(),"hooks");//存储GIT命令需要用的自定义脚本，默认禁用
}
```

2. Jit add 命令

- **代码逻辑**：向暂存区中添加文件，如果是文件类型就hash成blob对象，序列化后存储在.jit/objects中，同时在.jit\index中添加一行记录；如果是文件夹类型则遍历其下的每个文件并添加到暂存区中（首先要实现Blob类中各种的方法）
- **参数列表**：

参数	含义
repoPath	工作区路径
filename	添加的文件路径

Java

```java
public static void hash(String filename,String repoPath) throws Exception {
    String projPath = repoPath.replaceAll(".jit","");
    File file = new File(filename);
    //Blob hash
    if (file.isFile()) {
        Blob blob = new Blob(file);
        String hashRes = blob.getKey();
```

```
        String hashRes = blob.getKey();
        if (hashRes.length() != 40) {
            throw new IOException("hash value length error!");
        }
        String parentPath = blob.getPath();    //Blob类getPath()需要借助Repository类的加载
        File check_directory_exist = new File(parentPath + File.separator + hashRes.substring(0, 2));
        if (!check_directory_exist.exists()) {
            FileCreation.createDirectory(parentPath, hashRes.substring(0, 2));
        }
        File check_blob_exist = new File(parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2));
        if (!check_blob_exist.exists()) {

            FileCreation.createFile(parentPath + File.separator + hashRes.substring(0, 2), hashRes.substring(2), null);
            String path = parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2);

            blob.compressWrite(path, blob);    //对象序列化,压缩后写入.jit/objects
        }
        File index_file = new File(repoPath + File.separator + "index");
        if (!index_file.exists()) {
            FileCreation.createFile(repoPath, "index", null);
        }
        String relativePath = file.getPath().replaceAll(projPath.replace("\\", "\\\\"), "");
        String line = "100644" + " " + hashRes + " " + "0" + " " + relativePath + "\n";
        FileInputStream in = new FileInputStream(index_file);
        // size 为字串的长度 , 这里一次性读完
        int size = in.available();
        byte[] buffer = new byte[size];
        int flag = in.read(buffer);
        in.close();
        String str = new String(buffer, StandardCharsets.UTF_8);
        if (!str.contains(line)) {
            if (!str.contains(hashRes) && str.contains(relativePath)) {
                //add了一个修改过内容却没修改过文件名的处理方法
                //file.getPath()中有\,需要进行转义
                String filePath = relativePath.replace("\\", "\\\\");
                String regex = "\n.*?" + filePath;
                String newStr = str.replaceAll(regex, "");
                FileWriter fileWriter = new FileWriter(index_file);
                fileWriter.write(newStr);
                fileWriter.close();
            }
            FileWriter fileWriter = new FileWriter(index_file, true);
```

```
            fileWriter.write(line);
            fileWriter.close();
        }
    }
    //Tree hash
    if (file.isDirectory()) {
        File[] fs = file.listFiles();
        List<File> fileList = Tree.sortFile(fs);
        for (File f : fileList) {
            if (!f.getName().equals(".jit")){
                JitHash.hash(f.getPath(),repoPath);
            }
        }
    }
}
```

3. Jit commit 命令

- **代码逻辑**：提交当前暂存区所跟踪的文件，根据.jit\index中的文件记录，将.jit\objects对应的文件反序列化出来并组织成一个目录，然后hash成tree对象，序列化后存储在.jit/objects中；同时根据tree对象的hash值生成commit对象，序列化后写入.jit\objects，将当前HEAD指针指向该commit，而当前的commit则指向上一次commit（首先要实现Tree类和Commit类中各种的方法）
- **参数列表**：

参数	含义
repoPath	工作区路径
-m	
msg	commit的注释信息

Java

```
public static void commit(String repoPath, String message) throws Exception {
    FileInputStream in = new FileInputStream(repoPath + File.separator + "index");
    BufferedReader buffer = new BufferedReader(new InputStreamReader(in));
    String line;
    while((line = buffer.readLine()) != null) {
        String[] list = line.split(" ");
        Blob blob = Blob.deserialize(list[1]);
        //临时tree filePath: ../.jit/temp/相对路径
        String filePath = repoPath + File.separator + "temp" + File.separator + list[3];
        int index = filePath.lastIndexOf('\\');
```

```
        int index = filePath.lastIndexOf("\\");
        String dirPath = filePath.substring(0, index);
        String filename = filePath.substring(index+1);
        File file = new File(dirPath);
        if (!file.exists()) {
            file.mkdirs();
        }
        file = new File(dirPath, filename);
        if(!file.exists()) {
            file.createNewFile();
        }
        FileWriter fileWriter = new FileWriter(file);
        fileWriter.write(blob.getValue());
        fileWriter.close();
    }
    in.close();
    buffer.close();
    File file = new File(repoPath + File.separator + "temp");
    Tree tree = new Tree(file);
    String hashRes = tree.getKey();
    if (hashRes.length() != 40) {
        throw new IOException("hash value length error!");
    }
    String parentPath = tree.getPath();    //Tree类getPath()需要借助Repository类的加载
    File check_directory_exist = new File(parentPath + File.separator + hashRes.substring(0, 2));
    if (!check_directory_exist.exists()) {
        FileCreation.createDirectory(parentPath, hashRes.substring(0, 2));
    }
    File check_tree_exist = new File(parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2));
    if (!check_tree_exist.exists()) {
        FileCreation.createFile(parentPath + File.separator + hashRes.substring(0, 2), hashRes.substring(2), null);
        String path = parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2);
        tree.compressWrite(path, tree);    //对象序列化,压缩后写入.jit/objects
    }
    FileDeletion.deleteFile(file);
    InetAddress addr = InetAddress.getLocalHost();
    Date date = new Date();
    String hostname = addr.getHostName() + " " + date.getTime()/1000 + " +0800";
    Commit commit = new Commit(tree.getKey(), hostname, hostname, message);
    hashRes = commit.getKey();
    if (hashRes.length() != 40) {
        throw new IOException("hash value length error!");
    }
    check_directory_exist = new File(parentPath + File.separator +
```

```
hashRes.substring(0, 2));
    if (!check_directory_exist.exists()) {
        FileCreation.createDirectory(parentPath, hashRes.substring(0, 2));
    }
    File check_commit_exist = new File(parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2));
    if (!check_commit_exist.exists()) {
        FileCreation.createFile(parentPath + File.separator + hashRes.substring(0, 2), hashRes.substring(2), null);
        String path = parentPath + File.separator + hashRes.substring(0, 2) + File.separator + hashRes.substring(2);
        commit.compressWrite(path, commit);    //对象序列化,压缩后写入.jit/objects
    }
    // 第一次commit时创建main分支
    file = new File(repoPath + File.separator + "refs" +File.separator + "heads");
    if(file.list().length == 0) {
        FileCreation.createFile(file.getPath(), "main", hashRes);
    }
}
```

4. Jit rm 命令

- **代码逻辑**：从暂存区中删除文件，如果是文件类型就删除index文件里相应的行和.jit/objects目录下的相应blob；如果是文件夹则遍历其下的每个文件并删除
- **参数列表**：

参数	含义
repoPath	工作区路径
filename	删除的文件路径

Java

```
public static void remove(String repoPath, String filename) throws Exception {
    //定位工作区
    String workingPath=repoPath.replace(".jit","");
    if(!filename.contains(workingPath)){
        throw new IOException("filename is illegal");
    }
    if(!new File(filename).exists()){
        throw new IOException("file/directory does not exist");
    }
    if (new File(filename).isFile()){
        String relativePath=filename.replaceAll(workingPath.replace("\\","\\\\"),"");
```

```java
        //删除index文件里相应的行
        File index_file = new File(repoPath + File.separator + "index");
        FileInputStream in = new FileInputStream(index_file);
        // size 为字串的长度 ，这里一次性读完
        int size = in.available();
        byte[] buffer = new byte[size];
        int flag = in.read(buffer);
        in.close();
        String str = new String(buffer, StandardCharsets.UTF_8);
        //获取相对路径在Index中的位置以及blob对应的hash
        int file_index=str.lastIndexOf(relativePath);
        if (file_index!=-1){
            String file_hash=str.substring(file_index-43,file_index-3);
            String filePath = relativePath.replace("\\", "\\\\");
            String regex = "\n.*?" + filePath;
            String newStr = str.replaceAll(regex, "");
            //重写index文件
            FileWriter fileWriter = new FileWriter(index_file);
            fileWriter.write(newStr);
            fileWriter.close();
            //删除.jit/objects目录下的相应blob
            String dir=file_hash.substring(0,2);
            String file=file_hash.substring(2);
            String deletePath=repoPath+File.separator+"objects"+File.separator+dir+File.separator+file;
            FileDeletion.deleteFile(deletePath);
        }
    }
    //递归删除文件夹下的文件
    if(new File(filename).isDirectory()){
        File[] fs = new File(filename).listFiles();
        List<File> fileList = Tree.sortFile(fs);
        for (File f : fileList) {
            if (!f.getName().equals(".jit")){
                JitHash.remove(repoPath,f.getPath());
            }
        }
    }
}
```

5. Jit log 命令

- **代码逻辑**：从当前的HEAD指针指向的commit开始，依次向前遍历所有的commit信息，打印出当前commit的hash值，作者信息，commit时间戳，以及commit信息
- **参数列表**：

参数	含义
repoPath	工作区路径

Java

```java
public static void log() throws IOException, ClassNotFoundException {
    //取最新commit
    File HEAD = new File(Repository.getGitDir() + File.separator + "HEAD");
    String path = GitObject.getValue(HEAD).substring(5).replace("\r\n", "");
    File branchFile = new File(Repository.getGitDir() + File.separator + path);
    //path末尾有两个无效字符需要切掉
    String newest_commit=GitObject.getValue(branchFile).substring(0,40);
    //commit对象解压缩、反序列化
    Commit commit=Commit.deserialize(newest_commit);
    //用链表存成commit链
    LinkedList<Commit> commit_list=new LinkedList<>();
    commit_list.add(commit);
    if(commit.getParent()!=null){
        do {
        String next_commit=commit.getParent().substring(0,40);
        commit= Commit.deserialize(next_commit);
        commit_list.add(commit);

        } while (!Objects.equals(commit.getParent(), ""));
    }
    System.out.println("------------------------------jit log------------------------------");
    for(Commit c: commit_list){
        String[] list = c.getAuthor().split(" ");
        String author = list[0];
        Date date = new Date(Long.parseLong(list[1]));
        System.out.println("commit:  " + c.getKey());
        System.out.println("Author:  " + author);
        System.out.println("Date:  " + date + " " + list[2]);
        System.out.println("message:  " + c.getMessage());
        System.out.println("------------------------------------------------------------------------");
    }
}
```

6. Jit reset 命令

- **代码逻辑**：soft模式仅仅将HEAD指针指向指定的commit，其他不变；mixed模式在此基础上重置暂存区索引，根据commit指向的.jit\objects里对应文件反序列化出tree对象，使用tree的value值更新index文件；hard模式同时重置暂存区索引与工作区文件，根据commit指向的.jit\objects里

对应文件反序列化出tree对象，使用tree的treeList更新工作区（首先要实现recoverIndex与recoverWorkTree方法）

- **参数列表：**

参数	含义
repoPath	工作区路径
mode	回滚模式
commitId	commit的哈希值

Java

```java
public static void reset(String repoPath, String mode, String commitId) throws IOException, ClassNotFoundException {
    //检查commit对象是否存在
    Commit check_exist=Commit.deserialize(commitId);
    if(check_exist==null){
        throw new IOException("需要reset的commit对象不存在~~");
    }
    //获取当前分支指向的commit对象
    File HEAD = new File(repoPath + File.separator + "HEAD");
    String path = GitObject.getValue(HEAD).substring(5).replace("\r\n", "");
    File branchFile = new File(Repository.getGitDir() + File.separator + path);
    //重写commit指针
    FileWriter fileWriter = new FileWriter(branchFile);
    fileWriter.write(commitId);
    fileWriter.close();
    switch (mode){
        case "--soft" ->{
            break;
        }
        case "--mixed" ->{
            //重置index暂存区
            Commit commit = Commit.deserialize(commitId);
            Tree tree = Tree.deserialize(commit.getTree());
            File file = new File(repoPath + File.separator + "index");
            if(file.exists()) {
                file.delete();
            }
            file.createNewFile();
            recoverIndex(tree, "", file);
            break;
        }
        case "--hard" ->{
            //重置工作区
```

```
            Commit commit = Commit.deserialize(commitId);
            Tree tree = Tree.deserialize(commit.getTree());
            int index = repoPath.lastIndexOf('\\');
            String workDirectory = repoPath.substring(0, index);
            File file = new File(workDirectory);
            File[] fs = file.listFiles();
            for(File f : fs) {
                if(!f.getName().equals(".jit")) {
                    FileDeletion.deleteFile(f);
                }
            }
            recoverWorkTree(tree, workDirectory);
        }
    }
}

public static void recoverIndex(Tree t, String parentTree, File indexFile) throws IOException {
    ArrayList<String> list = FileReader.readByBufferReader(t.getValue());
    ArrayList<GitObject> treeList = t.getTreeList();
    Iterator iterator = treeList.iterator();
    boolean isRootDir = true;
    for (String s : list) {
        if (FileReader.readObjectFmt(s).equals("blob")) {
            Blob blob = (Blob)iterator.next();
            String fileName = FileReader.readObjectFileName(s);
            String filePath = parentTree + File.separator + fileName;
            String line = "100644" + " " + blob.getKey() + " " + "0" + " " + filePath + "\n";
            FileWriter fileWriter = new FileWriter(indexFile, true);
            fileWriter.write(line);
            fileWriter.close();
        }
        else {
            String dirName = FileReader.readObjectFileName(s);
            if(isRootDir) {
                isRootDir = false;
            }
            else {
                Tree tree = (Tree)iterator.next();
                if(parentTree.equals("")) {
                    recoverIndex(tree, dirName, indexFile);
                }
                else {
                    recoverIndex(tree, parentTree + File.separator + dirName, indexFile);
                }
            }
```

```
        }
    }
}

public static void recoverWorkTree(Tree t, String parentTree) throws IOException {
    ArrayList<String> list = FileReader.readByBufferReader(t.getValue());
    ArrayList<GitObject> treeList = t.getTreeList();
    Iterator iterator = treeList.iterator();
    boolean isRootDir = true;
    for (String s : list) {
        if (FileReader.readObjectFmt(s).equals("blob")) {
            String fileName = FileReader.readObjectFileName(s);
            Blob blob = (Blob)iterator.next();
            FileCreation.createFile(parentTree, fileName, blob.getValue());
        }
        else {
            String dirName = FileReader.readObjectFileName(s);
            if(isRootDir) {
                isRootDir = false;
            }
            else {
                Tree tree = (Tree)iterator.next();
                FileCreation.createDirectory(parentTree, dirName);
                recoverWorkTree(tree, parentTree + File.separator + dirName);
            }
        }
    }
}
```

7. Jit branch 命令

- **代码逻辑**：.jit\refs\heads目录下的文件即为分支，打印文件名、新建和删除文件对应branch的三种操作，如果要删除的分支与HEAD指向的分支相同，则抛出异常
- **参数列表**：

参数	含义
repoPath	工作区路径
mode	可选参数，NULL打印或新建分支，-d删除分支
branchName	可选参数，NULL打印分支，非空新建或删除名为*branchName*的分支

Java

```
public static void branch(String repoPath) throws IOException {
```

```
    File HEAD = new File(Repository.getGitDir() + File.separator + "HEAD");
    String path = GitObject.getValue(HEAD).substring(5).replace("\r\n", "");
    int index = path.lastIndexOf('/');
    String head = path.substring(index+1);
    File file = new File(repoPath + File.separator + "refs" + File.separator + "heads");
    if(file.isDirectory()) {
        File[] fs = file.listFiles();
        for(File f : fs) {
            if(f.getName().equals(head)) {
                System.out.println("* " + head);
            }
            else {
                System.out.println("  " + f.getName());
            }
        }
    }
}

public static void createBranch(String repoPath, String branchName) throws IOException {
    //取当前commit
    File HEAD = new File(Repository.getGitDir() + File.separator + "HEAD");
    String path = GitObject.getValue(HEAD).substring(5).replace("\r\n", "");
    File branchFile = new File(Repository.getGitDir() + File.separator + path);
    //path末尾有两个无效字符需要切掉
    String current_commit=GitObject.getValue(branchFile).substring(0,40);
    File file = new File(repoPath + File.separator + "refs" + File.separator + "heads" + File.separator + branchName);
    if(!file.exists()) {
        file.createNewFile();
    }
    FileWriter fileWriter = new FileWriter(file);
    fileWriter.write(current_commit);
    fileWriter.close();
}

public static void deleteBranch(String repoPath, String branchName) throws Exception {
    File HEAD = new File(Repository.getGitDir() + File.separator + "HEAD");
    String path = GitObject.getValue(HEAD).substring(5).replace("\r\n", "");
    int index = path.lastIndexOf('/');
    String head = path.substring(index+1);
    if(branchName.equals(head)) {
        throw new IllegalArgumentException("error: Cannot delete branch " + branchName);
    }
    else {
        FileDeletion.deleteFile(repoPath + File.separator + "refs" +
```

```
File.separator + "heads" + File.separator + branchName);
    }
}
```

8. Jit checkout 命令

- **代码逻辑**：修改HEAD指针指向的文件即可实现分支切换
- **参数列表**：

参数	含义
repoPath	工作区路径
mode	可选参数，NULL切换分支，-b新建分支后再切换
branchName	分支名

Java

```java
public static void checkout(String repoPath, String branchName) throws IOException, ClassNotFoundException {
    File file = new File(repoPath + File.separator + "refs" + File.separator + "heads" + File.separator + branchName);
    if(!file.exists()) {
        throw new IOException("需要checkout的分支不存在~~");
    }
    //path末尾有两个无效字符需要切掉
    String current_commit=GitObject.getValue(file).substring(0,40);
    //更新head
    FileDeletion.deleteFile(repoPath + File.separator + "HEAD");
    FileCreation.createFile(repoPath,"HEAD","ref: refs/heads/"+branchName);//创建“HEAD”文件，存储当前HEAD指向的分支
    //更新工作区
    JitHash.reset(repoPath, "--hard", current_commit);
}
```

2.3 测试

Java

```java
jit init D:\test1
```

```
文件夹创建成功： D:\test1\.jit
文件创建成功： D:\test1\.jit\COMMIT_EDITMSG
文件创建成功： D:\test1\.jit\config
文件创建成功： D:\test1\.jit\HEAD
文件创建成功： D:\test1\.jit\description
文件夹创建成功： D:\test1\.jit\objects\info
文件夹创建成功： D:\test1\.jit\objects\pack
文件夹创建成功： D:\test1\.jit\refs\heads
文件夹创建成功： D:\test1\.jit\refs\tags
文件夹创建成功： D:\test1\.jit\logs
文件夹创建成功： D:\test1\.jit\info
文件夹创建成功： D:\test1\.jit\hooks
Jit repository has been initiated successfully.
```

Java

```java
jit add D:\test1 D:\test1\test1.1
jit rm D:\test1 D:\test1\test1.1
```

```
文件夹创建成功： D:\test1\.jit\objects\e3
文件创建成功： D:\test1\.jit\objects\e3\a5e6bba3587656c65c34d79174165965c6dfea
文件创建成功： D:\test1\.jit\index
文件夹创建成功： D:\test1\.jit\objects\5a
文件创建成功： D:\test1\.jit\objects\5a\430ff389804157b6cac769124350a65ace42ee
文件夹创建成功： D:\test1\.jit\objects\f7
文件创建成功： D:\test1\.jit\objects\f7\88a0f4533294286a102438b1d939d944d2be87
文件夹创建成功： D:\test1\.jit\objects\ae
文件创建成功： D:\test1\.jit\objects\ae\340e4751368f3f7831ba8ff98f0b11bbf4f3fa
文件夹创建成功： D:\test1\.jit\objects\0f
文件创建成功： D:\test1\.jit\objects\0f\70e0221addd7bb6a271c36fc7d0827d8b1f12c
文件夹创建成功： D:\test1\.jit\objects\34
```

Java

```java
jit commit D:\test1 -m test1
```

```
文件夹创建成功： D:\test1\.jit\objects\ec
文件创建成功： D:\test1\.jit\objects\ec\30776c30f79d421f3172ed7b1a67bf51e11770
文件夹创建成功： D:\test1\.jit\objects\38
文件创建成功： D:\test1\.jit\objects\38\44313b03ab1f1e3eeaa789ca29834860a8a2e5
```

Java

```java
jit log D:\test1
```

```
-----------------------------jit log----------------------------------
commit:  65d4e3a61e32703e102297a5560e0bc278a19d2f
Author:  DESKTOP-5SDOJMI
Date:  Tue Jan 20 07:56:09 CST 1970 +0800
message:  test3
----------------------------------------------------------------------
commit:  b04a7ee344265b8059ce03e99a9ce8d011fdbf64
Author:  DESKTOP-5SDOJMI
Date:  Tue Jan 20 07:56:09 CST 1970 +0800
message:  test2
----------------------------------------------------------------------
commit:  3844313b03ab1f1e3eeaa789ca29834860a8a2e5
Author:  DESKTOP-5SDOJMI
Date:  Tue Jan 20 07:56:08 CST 1970 +0800
message:  test1
----------------------------------------------------------------------
```

Java

```
jit reset D:\test1 --soft 65d4e3a61e32703e102297a5560e0bc278a19d2f
```

main - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
65d4e3a61e32703e102297a5560e0bc278a19d2f
```

Java

```
jit reset D:\test1 --mixed 65d4e3a61e32703e102297a5560e0bc278a19d2f
```

index - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
100644 e3a5e6bba3587656c65c34d79174165965c6dfea 0 test1.1\test\test.txt
100644 5a430ff389804157b6cac769124350a65ace42ee 0 test1.1\Chapter1.html
100644 f788a0f4533294286a102438b1d939d944d2be87 0 test1.1\Chapter10.html
100644 ae340e4751368f3f7831ba8ff98f0b11bbf4f3fa 0 test1.1\Chapter2.html
100644 0f70e0221addd7bb6a271c36fc7d0827d8b1f12c 0 test1.1\Chapter3.html
100644 342eb39c1d0e942f16c6c074ce79e6c8715219ed 0 test1.1\Chapter4.html
100644 ff75a30c0396978640142c7edab70932baf47031 0 test1.1\Chapter5.html
100644 8dbe4d49059c083144f5797f86466a67dce8b4db 0 test1.1\Chapter6.html
100644 90719db610e3ff79c91cd4f15891db61282a231d 0 test1.1\Chapter7.html
100644 0e9d425860dbe9a66c2d6c679cbdb468a4aeb70a 0 test1.1\Chapter8.html
100644 893f1ce57fc4251e2fc9752b09322cc45cbc25ee 0 test1.1\Chapter9.html
100644 1cfdacbaa0f47afdd0ab1c557077930e0395603c 0 test1.2\P9.pptx
100644 f8dcd858c91451518e18d62cdc6c70cb99671fda 0 test1.2\新实验手册P9.pdf
100644 a2c7ef0eae7b89b4529ad32fe35ce4e7d6c7b6bf 0 test1.2\闯关秘籍P9.docx
100644 959b974b43304ff6b02b1759f2503f05cb7063cf 0 test1.3\0_0课程介绍.pdf
100644 4b1588bf5927a08d8805003515c25dd4fc5ae63b 0 test1.3\0_1第一讲：python简介.pdf
100644 786e2575ed08e933623b5b81566a486cc055e92d 0 test1.3\0_2第二讲：python基础语法.pdf
100644 817c5d2c8f5fa6c1dcba9c31d144c8f21e29afaf 0 test1.3\0_3第三讲：函数.pdf
100644 391e1c73ac4233a298a3d485e8d14e1e629ddb4d 0 test1.3\0_5第五讲：IO编程与异常.pdf
100644 04a18fae65b346f855a95b10a1e33684c3789bb4 0 test1.3\elevator.vsdx
100644 f2f8d1f818caedb4db7fca7af7ead6efb67d7e53 0 test1.3\python-io文件操作.ipynb
100644 180de89549883262c63e73f7311254f4175e8990 0 test1.3\随机密码生成.mdj
100644 744b3c3e9d6cc47ca55cb1e000ac0cece7912aae 0 test1.3\随机密码生成.png
```

Java

```
jit reset D:\test1 --hard 65d4e3a61e32703e102297a5560e0bc278a19d2f
```

```
文件夹创建成功： D:\test1\test1.1
文件夹创建成功： D:\test1\test1.1\test
文件创建成功： D:\test1\test1.1\test\test.txt
文件创建成功： D:\test1\test1.1\Chapter1.html
文件创建成功： D:\test1\test1.1\Chapter10.html
文件创建成功： D:\test1\test1.1\Chapter2.html
文件创建成功： D:\test1\test1.1\Chapter3.html
文件创建成功： D:\test1\test1.1\Chapter4.html
文件创建成功： D:\test1\test1.1\Chapter5.html
文件创建成功： D:\test1\test1.1\Chapter6.html
文件创建成功： D:\test1\test1.1\Chapter7.html
文件创建成功： D:\test1\test1.1\Chapter8.html
文件创建成功： D:\test1\test1.1\Chapter9.html
文件夹创建成功： D:\test1\test1.2
```

Java

```java
jit branch D:\test1 newbranch
jit branch D:\test1
```

```
* main
  newbranch
```

Java

```java
jit checkout D:\test1 newbranch
```

```
文件夹创建成功： D:\test1\test1.1
文件夹创建成功： D:\test1\test1.1\test
文件创建成功： D:\test1\test1.1\test\test.txt
文件创建成功： D:\test1\test1.1\Chapter1.html
文件创建成功： D:\test1\test1.1\Chapter10.html
文件创建成功： D:\test1\test1.1\Chapter2.html
文件创建成功： D:\test1\test1.1\Chapter3.html
文件创建成功： D:\test1\test1.1\Chapter4.html
文件创建成功： D:\test1\test1.1\Chapter5.html
文件创建成功： D:\test1\test1.1\Chapter6.html
文件创建成功： D:\test1\test1.1\Chapter7.html
文件创建成功： D:\test1\test1.1\Chapter8.html
文件创建成功： D:\test1\test1.1\Chapter9.html
文件夹创建成功： D:\test1\test1.2
```

```
  main
* newbranch
```

Java

```java
jit checkout D:\test1 -b master
```

```
  main
* master
  newbranch
```

Java

```
jit branch D:\test1 -d main
```

```
* master
  newbranch
```

三、项目心得

> - 在深入理解git原理的基础上，对init、add、commit、log、remove、reset、branch、checkout命令进行了基本实现。
> - 在实践中对git版本控制系统中的三种基本对象blob、tree、commit的实现方式与键值对存储原理有了更深一步认识
> - git的底层思想根本上是记录文件的版本变更，掌握如何记录变更的方法才是真正理解git的主线。git通过blob对象存储各版本文件内容、通过tree对象解决文件名保存和目录组织问题、每一次变更存储成一个tree对象并指向一次commit对象，保证了每次的版本变更可追溯、可还原等。