컴퓨터학부 20201841 박세연

1. 과제개요

ssu\_repo 프로그램은 사용자가 파일과 디렉터리의 변경사항을 추적하고 관리할 수 있도록 도와주는 버전 관리 시스템이다. 이 프로그램은 사용자가 지정한 디렉터리에 있는 파일들을 상태 관리 대상으로 추가하거나 제거함으로써, 파일의 변경 내역을 추적하고, 이러한 변경사항을 백업 및 복원하며, 버전을 관리할 수 있도록 설계되었다. 주요 명령어로는 add, remove, status, commit, revert, log, help, exit 등이 있다.

add는 스테이징 구역에 추가하여 변경 내용을 추적할 경로를 인자로 입력받고, 해당 파일 또는 디렉터리 경로를 스테이징 구역에 추가한다. 인자로 파일이 입력되었다면 해당 파일을 스테이징 구역에 추가하고, 디렉터리가 입력되었다면 입력받은 디렉터리 내의 파일들과 서브 디렉터리 내의 파일들까지 모두 재귀적으로 스테이징 구역에 추가시킨다.

remove는 스테이징 구역에서 삭제하여 더 이상 변경내역을 추적하지 않을 경로를 입력받아 해당 파일 혹은 디렉터리 경로를 스테이징 구역에서 제거하도록 하는 명령어이다. 인자로 들어온 파일이 스테이징 구역에 있는지 확인하고 스테이징 구역에 존재한다면 remove를 성공시킨다. 해당 파일이 스테이징 구역에 존재하지 않는다면 이 파일이 add후에 remove되어 존재하지 않는 것인지 혹은 add 조차 되지 않아 스테이징 구역에 존재하지 않는 것인지를 판단한다. remove가 되어 존재하지 않는 경우에는 이미 삭제되었음을 출력하여 알리고, add조차 된 적이 없어 스테이징 구역에 존재하지 않는 것이라면 remove를 성공시킨다.

status는 스테이징 구역에 추가된 경로들의 파일들에 대해 변경 내역을 추적하여 정보를 출력하는 명령어이다. 현재 작업 경로 내에 존재하는 파일들을 대상으로 탐색하며 스테이징 구역에 포함되어 있지 않은 파일들은 Untracked files에 출력한다. 현재 작업 경로 내에 파일들을 대상으로 스테이징 구역에 포함되어있는 경로의 파일들에 대해서는, 마지막 백업 파일을 기준으로 변경되었거나 삭제된 것에 대해서는 Changes to be committed에 파일 상태와 경로를 함께 출력한다.

commit은 명령어에서 커밋 이름을 함께 입력받아 해당 이름으로 버전 디렉터리를 생성하여 스테이징 구역에 포함되어 있는 경로 내 파일들에 대해 마지막 백업 파일에서 변경되거나 삭제되거나 새롭게 추가된 파일들에 대해서 백업을 진행하는 명령어이다.

revert는 커밋 이름을 입력받아서 해당 이름의 백업 파일 버전으로 원본을 되돌리는 명령어이다.

log는 커밋 기록에 대한 로그를 출력하는 명령어이다. 인자를 생략하면 커밋 로그 파일에 있는 모든 커밋 로그를 커밋 이름 별로 출력한다. 인자를 입력한 경우, 인자에 해당하는 커밋 이름을 가진 로그를 출력해준다. 만약 존재하지 않는 커밋 이름을 입력한 경우에는 예외처리를 했다.

help는 위 프로그램에서 사용할 수 있는 모든 명령어에 대한 설명을 출력해주는 명령어이다. help 뒤에 명령어 인자를 넘겨주지 않으면 전체 명령어에 대한 usage를 볼 수 있으며, 인자로 특정 명령어를 입력한다면 해당 명령어의 usage만 단독으로 출력한다.

exit 명령어는 실행 중인 ssu\_repo 프로그램을 종료한다. 사용자는 exit을 입력하기 전까지 계속해서 프로그램을 사용할 수 있고 exit 명령어를 통해 프로그램을 종료한다.

2. 구현 기능

**<ssu\_header.h>**

char exeNAME[PATHMAX];

char exePATH[PATHMAX];

char homePATH[PATHMAX];

char repoPATH[PATHMAX];

char commitlogPATH[PATHMAX]; // 커밋 로그 파일 경로

char staginglogPATH[PATHMAX]; // 스테이징 구역 파일 경로

int hash;

// 명령어 이름 저장한 배열

char \*commanddata[10] = {

"add",

"remove",

"status",

"commit",

"revert",

"log",

"help",

"exit"};

// 받아온 명령어들을 나누어 저장할 구조체

typedef struct command\_parameter

{

char \*command;

char \*filename;

char \*tmpname;

int commandopt;

char \*argv[256];

} command\_parameter;

typedef struct \_fileNode

{

struct \_dirNode \*root\_dir; // 파일노드가 속한 상위 디렉터리를 가리킴

char file\_path[PATHMAX]; // 파일 경로 저장

struct \_fileNode \*prev\_file; // 이전 파일 노드

struct \_fileNode \*next\_file; // 다음 파일 노드

} fileNode;

typedef struct \_dirNode

{

struct \_dirNode \*root\_dir; // 디렉터리 노드가 속한 상위 디렉터리

char dir\_path[PATHMAX]; // 디렉터리 경로

fileNode \*file\_head; // 디렉터리에 속한 파일들을 관리하는 연결리스트의 헤드를 가리킴

struct \_dirNode \*subdir\_head; // 하위 디렉터리들을 관리하는 연결 리스트의 헤드를 가리킴

struct \_dirNode \*prev\_dir; // 이전 디렉터리 노드

struct \_dirNode \*next\_dir; // 다음 디렉터리 노드

} dirNode;

dirNode \*staging\_dir\_list; // staging된 파일 및 디렉터리들을 관리하기 위한 최상위 디렉터리 역할

dirNode \*managing\_dir\_list; // add나 remove를 한번이라도 하면 추가되는 리스트 노드 (삭제되지 않음)

dirNode \*commit\_dir\_list; // commit하여 백업한 노드들을 저장할 리스트 노드

// 디렉터리 경로를 저장하기 위한 연결리스트 노드

typedef struct pathList\_

{

struct pathList\_ \*next;

struct pathList\_ \*prev;

char path[NAMEMAX];

} pathList;

// 커밋 로그 저장 구조체

typedef struct CommitLogEntry

{

char commitMessage[256];

char changeType[32]; // "new file", "modified", "removed"

char filePath[PATHMAX];

char backupPath[PATHMAX];

struct CommitLogEntry \*next; // 다음 노드를 가리키는 포인터

} CommitLogEntry;

CommitLogEntry \*commitHead = NULL;

CommitLogEntry \*appendEntry(CommitLogEntry \*\*head, CommitLogEntry \*newEntry);

// 커밋 로그 노드 삽입 함수

int parseCommitLog(const char \*line, CommitLogEntry \*entry); // 커밋로그에서 파싱하는 함수

void printCommitLogEntry(const CommitLogEntry \*entry); // 커밋 로그 구조체 값 확인 출력 함수

void printCommitLogEntryGrouped(CommitLogEntry \*head); // 로그를 커밋메세지 별로 출력하는 함수

void printSpecificCommitLog(CommitLogEntry \*head, const char \*commitMessage);

// 특정 커밋 메세지에 대한 로그만 출력하고 없으면 에러 처리

void testStagingList(dirNode \*stagingList); // 연결리스트 구축 확인용 코드

int md5(char \*target\_path, char \*hash\_result); // 파일 내용을 읽어 MD5 해시 값을 계산하고 hash\_result에 저장

int ConvertHash(char \*target\_path, char \*hash\_result); // 해시 실행

int cmpHash(char \*path1, char \*path2); // path1과 path2의 해시 값을 비교하는 함수

char \*cvtNumComma(int a); // 3자리 수씩 끊어서 문자열로 변환하는 함수

char \*GetFileName(char file\_path[]);// 경로에서 파일 이름만을 추출하는 함수

char \*strToHex(char \*str); // str을 입력받아 16진수 형태의 문자열로 변환 후 반환

char \*getDate(); // 일정한 형식으로 일시를 받아오는 함수

char \*QuoteCheck(char \*\*str, char del); // 문자열에서 구분자를 찾아서 구분자를 제거하는 함수

char \*Tokenize(char \*str, char \*del); // 구분자를 기준으로 토큰을 분리하는 함수

char \*\*GetSubstring(char \*str, int \*cnt, char \*del); // Tokenize 호출함

int ConvertPath(char \*origin, char \*resolved); // 절대 경로로 변환하는 함수

int cmpPath(char \*path1, char \*path2); // 두 경로를 비교하여 정렬 순서를 결정

char \*substr(char \*str, int beg, int end); // 주어진 문자열인 str에서 특정 범위의 문자열을 추출

char \*c\_str(char \*str); // 주어진 문자열을 복사하여 새로운 문자열 생성(입력된 문자열의 복사본 반환)

void fileNode\_init(fileNode \*\*file\_node); // 파일 노드 초기화 함수

void dirNode\_init(dirNode \*\*node); // 디렉터리 노드 초기화 함수

fileNode \*find\_file\_node(fileNode \*head, const char \*path);

// 연결리스트 구축 과정에서 : 중복된 파일 노드를 거르기 위한 함수

dirNode \*find\_directory\_node(dirNode \*head, const char \*path);

// 연결리스트 구축 과정에서 : 중복된 디렉터리 노드를 거르기 위한 함수

void insert\_directory\_node(dirNode \*\*head, dirNode \*parent, const char \*path); // 디렉터리 노드 삽입 함수

void insert\_file\_node(dirNode \*parent, const char \*path); // 파일 노드 삽입 함수

void remove\_file\_node(dirNode \*parent, const char \*path); // 파일 노드 삭제 함수

void remove\_dir\_node(dirNode \*\*head, dirNode \*nodeToDelete);

// 연결리스트에서 디렉터리 노드를 삭제하는 함수

void recursiveDeleteEmptyDirs(dirNode \*\*head);

// 루트 디렉터리부터 하위 디렉터리를 재귀적으로 탐색하며 비어있는 디렉터리 삭제

void add\_staging(char \*path, dirNode \*root);

// add된 로그 파일의 경로를 읽어 연결리스트 구축 (경로들을 한 줄씩 인자로 받음)

void remove\_staging(char \*path, dirNode \*root);

// remove된 로그 파일의 경로를 읽어 연결리스트 구축 (경로들을 한 줄씩 인자로 받음)

bool is\_fully\_staged\_directory(dirNode \*root, const char \*dirPath);

// 경로로 디렉터리가 들어왔을때 디렉터리 내의 모든 파일이나 디렉터리들이 staging 되었는지 확인하는 함수

fileNode \*find\_file\_recursively(dirNode \*dir, const char \*path);

// 재귀적으로 탐색하며 해당 파일이 staging 되었는지 확인하는 함수

bool is\_staged\_file\_or\_directory(dirNode \*root, const char \*path, struct stat \*statbuf);

// 경로로 들어온 파일 혹은 디렉터리가 이미 staging되어있는지 확인하기 위한 함수

bool is\_none\_staged\_in\_directory(dirNode \*root, const char \*dirPath);

// 하나라도 있으면 false반환, 모두 없어야 true 반환

bool is\_none\_file\_or\_directory(dirNode \*root, const char \*path, struct stat \*statbuf);

// 경로로 디렉터리가 들어온 경우 하나라도 있으면 false 반환, 모두 존재하지 않는 경우 true 반환 / 파일의 경우 존재하면 false 반환

int isInStagingList(const char \*path);

// 주어진 경로가 staging 리스트에 존재하지 않으면 false 반환, 존재하면 true 반환

int isInManagingList(const char \*path);

// 주어진 경로가 managing 리스트에 존재하지 않으면 false 반환, 존재하면 true 반환

int statusUntrackedCheck(const char \*basePath, int \*firstPrint);

// 재귀적으로 디렉토리를 탐색하며 파일 상태를 출력하는 함수

**<ssu\_backup.c>**

int AddCommand(command\_parameter \*parameter); // Add 명령어를 실행하는 함수

int RemoveCommand(command\_parameter \*parameter); // remove 명령어를 실행하는 함수

int StatusCommand(command\_parameter \*parameter); // status 명령어를 실행하는 함수

int CommitCommand(command\_parameter \*parameter); // commit 명령어를 실행하는 함수

int LogCommand(command\_parameter \*parameter); // log 명령어를 실행하는 함수

void CommandFun(char \*\*arglist); // 입력된 명령어에 따라 명령 함수 실행

void CommandExec(command\_parameter parameter); // 명령어 실행 함수

void HelpExec(char \*command); // help 실행 함수

void ParameterInit(command\_parameter \*parameter); // 파라미터 초기화 함수

int ParameterProcessing(int argcnt, char \*\*arglist, int command, command\_parameter \*parameter);

// 명령줄로 받은 명령어들을 정리하고 알맞게 저장하는 함수

int Prompt(); // 사용자에게 명령어를 계속 입력받기 위해 호출하는 함수

int init\_staging\_list(int log\_fd);

// staging 로그 파일을 읽어서 staging\_dir\_list와 managing\_dir\_list 연결리스트를 구축하는 함수

int init\_commit\_list(); // commit 로그 파일을 읽어서 커밋 구조체 리스트를 구축하는 함수

void Init(); // 초기화 함수

**<ssu\_help.c>**

// help 명령어를 실행하는 함수로 command 인자로 출력하려는 명령어 받음

// 사용자가 인자를 생략할 경우 임의로 “all” 문자열을 command로 받아 모든 명령어에 대한 usage 출력

void help(char \*command);

void helpAdd(); // add 명령어에 대한 usage를 출력하는 함수

void helpRemove(); // remove 명령어에 대한 usage를 출력하는 함수

void helpStatus(); // status 명령어에 대한 usage를 출력하는 함수

void helpCommit(); // commit 명령어에 대한 usage를 출력하는 함수

void helpRevert(); // revert 명령어에 대한 usage를 출력하는 함수

void helpLog(); // log 명령어에 대한 usage를 출력하는 함수

void helpHelp(); // help 명령어에 대한 usage를 출력하는 함수

void helpExit(); // exit 명령어에 대한 usage를 출력하는 함수

3. 상세 설계

**<메인 흐름도>**

도표, 스케치, 그림, 패턴이(가) 표시된 사진

자동 생성된 설명

**<add 명령어 – AddCommand() 흐름도>**

도표, 텍스트, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

**<remove 명령어 – RemoveCommand() 흐름도>**

**도표, 라인, 화이트, 스케치이(가) 표시된 사진

자동 생성된 설명**

**<status 명령어 – StatusCommand() 흐름도>**

도표, 라인, 화이트, 스케치이(가) 표시된 사진

자동 생성된 설명

**<log 명령어 – LogCommand() 흐름도>**

도표, 라인, 스크린샷, 스케치이(가) 표시된 사진

자동 생성된 설명

4. 실행결과

**(1) ssu\_repo**

스크린샷, 텍스트, 소프트웨어, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

- ssu\_repo 프로그램 실행 시 레포 디렉터리 생성, 하위에 커밋 로그와 스테이징 구역 생성

- 엔터 입력 시 프롬프트 재출력

- 지정한 명령어 이외의 명령어 입력 시 help 명령어의 결과 출력

**(2) 내장 명령어 1. add**

텍스트, 소프트웨어, 컴퓨터, 웹 페이지이(가) 표시된 사진

자동 생성된 설명

- 경로 입력하지 않을 경우, add 명령어에 대한 에러 처리 및 Usage 출력 후 프롬프트 재출력

- 존재하지 않는 경로를 입력할 경우, 에러 출력 후 프롬프트 재출력

- 파일 및 디렉터리가 모두 잘 스테이징 구역에 올라가고 있음

**(3) 내장 명령어 2. remove**

텍스트, 소프트웨어, 컴퓨터, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

- 경로 입력하지 않을 경우, remove 명령어에 대한 에러 처리 및 Usage 출력 후 프롬프트 재출력

- 존재하지 않는 경로를 입력할 경우, 에러 출력 후 프롬프트 재출력

- 파일 및 디렉터리를 스테이징 구역에서 잘 제거하고, 다시 add하고 remove할 때도 잘 진행되고 있음

**(4) 내장 명령어 3. status**

- commit 명령어를 구현하지 않았기 때문에 커밋 로그 파일이 자동적으로 작성이 되지 않아 commit 파일을 임의로 작성하고, 백업 파일도 생성되지 않기 때문에 임의로 생성한 후에 명령어를 실행한다.

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

자동 생성된 설명

- add 와 remove 명령어를 이용해서 스테이징 구역의 파일 내용을 명세와 같게 했고, 커밋 로그 파일를 임의로 작성했고, 커밋으로 인해 생성되는 백업 디렉터리(/home/ubuntu/lsp/.repo/first commit)와 백업 파일들도 임의로 생성해주었다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

- 왼쪽은 원본 파일(/home/ubuntu/lsp/a.txt)이고 오른쪽은 임의로 생성한 백업된 파일(/home/ubuntu/lsp/.repo/first commit/.a.txt)이다. 아직은 백업 후 원본에 대한 수정이나 삭제가 이루어지지 않은 시점이기 때문에 같은 내용을 파일이다.

텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

- 왼쪽은 원본 파일(/home/ubuntu/lsp/a/a.txt)이고 오른쪽은 임의로 생성한 백업된 파일(/home/ubuntu/lsp/.repo/first commit/.a/a.txt)이다. 아직은 백업 후 원본에 대한 수정이나 삭제가 이루어지지 않은 시점이기 때문에 같은 내용을 파일이다.

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

- ./a/b/c.txt 파일의 내용을 변경하고, ./a/c.txt 파일을 새로 생성했다.

- status 명령어를 실행하여, 추적하고 있는 파일들 중에서는 변경된 것이 없기 때문에 Nothing to commit이 출력된다.

- ./a/b/c.txt 파일을 변경했지만, 이미 위에서 추적하지 않도록 remove 했기 때문에 변경을 추적하지 않는다.

- ./a/c.txt 파일을 추가하고 아직 add 하지 않아 추적하지 않기 때문에 Untracked files에 출력된다.

- ./a.txt 파일의 내용을 변경하고, ./a/a.txt 파일을 삭제했다.

- status 명령어를 실행하여, 추적하고 있는 파일들 중 삭제되거나 변경된 파일들을 출력한다.

- ./a.txt 파일의 내용이 수정되었기 때문에 modified 파일 상태와 경로를 출력한다.

- ./a/a.txt 파일을 삭제했기 때문에 removed 파일 상태와 경로를 출력한다.

**(5) 내장 명령어 6. log**

- commit 명령어를 구현하지 않았기 때문에 커밋 로그 파일이 자동적으로 작성이 되지 않아 commit 파일을 임의로 작성한 후에 명령어를 실행한다.

텍스트, 소프트웨어, 스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

- 인자 생략 시, 모든 커밋 로그를 커밋 이름 별로 출력

- 인자 지정시, 해당 커밋에 대한 로그만 출력

- 존재하지 않는 커밋 이름을 인자로 입력한 경우 에러 처리후 프롬프트 재출력

**(6) 내장 명령어 7. help**

**텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명**

- help 명령어 실행 시 내장 명령어에 대한 Usage 출력

- 인자 생략 시, 모든 명령어에 대한 Usage 출력

- 인자 지정 시, 해당 명령어에 대한 Usage 출력

- 잘못된 내장 명령어 입력 시 에러 처리 후 프롬프트 재출력

**(7) 내장 명령어 8. exit**

**텍스트, 스크린샷, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명**

- exit 명령어 입력 시 실행 중인 ssu\_repo 프로그램 종료