컴퓨터학부 20202920 조민혁

1.과제개요

명령어를 이용하여 파일을 백업하고, 백업파일을 삭제하며, 백업파일을 통해서 복구할 수 있고, 사용가능한 명령어들의 종류를 출력할 수 있는 프로그램입니다. Backup기능에는 -d,-r,-y의 옵션을 사용할 수 있고, remove는 -d,-r,-a의 옵션을 사용할 수 있으며, recover기능에는 -d,-r,-l의 옵션을 사용할 수 있습니다. 또한 help명령어를 통해 명령어들의 종류를 확인할 수 있습니다. help명령어만 입력하였을 때는 모든 명령어에 대한 설명이 나오고 help 명령어 뒤에 다른 명령어들을 입력하면 각각의 내장명령어에 대한 설명을 출력할 수 있습니다.

2.기능

backup : path를 입력받으면 해당 path에 존재하는 파일에 대하여 /home/backup디렉토리안의 현재시간을 나타내는 디렉토리를 생성하여 그 안에 path에 존재하는 파일을 백업합니다. 같은 내용의 백업파일이 있을 경우 백업을 진행하지 않습니다.

-d 옵션: 파일의 경로 대신에 디렉토리 경로를 집어넣어서 디렉토리 안에 있는 디렉토리가 아닌 파일들을 /home/backup디렉토리안의 현재시간을 나타내는 디렉토리를 생성하여 그 안에 넣어서 백업합니다.

-r 옵션: 디렉토리 경로를 입력받아서 BFS를 재귀적으로 사용하여 디렉토리 내부의 모든 파일들과 하부 디렉토리의 파일들까지 backup합니다.

-y 옵션 : 기존의 백업이 같은 내용의 백업파일이 있을 경우 백업을 진행하지 않는데 이 옵션을 사용하면 같은 내용의 백업파일이 있는 경우에도 백업을 진행합니다.

remove: /home/backup 디렉토리 밑에 있는 backup명령어를 통해 생성한 백업파일들을 선택적으로 삭제할 수 있습니다. 한 개만 있는 경우에는 자동적으로 그 한 개의 파일이 삭제되지만, 파일이 여러 개인 경우 선택하여 삭제할 수 있습니다.

-d 옵션 : 파일 경로 대신에 디렉토리 경로를 집어넣어 해당 디렉토리 안에 있는 디렉토리가 아닌 파일들을 /home/backup/12자리의 시각 디렉토리 안의 파일들을 전부 삭제합니다.

-r 옵션 : 디렉토리 경로를 입력받아서 BFS를 재귀적으로 사용하여 디렉토리 내부의 모든 파일들과 하부 디렉토리의 파일들까지 remove 합니다.

-a 옵션 : 하나의 경로에 여러 개의 백업본이 생길 수 있기 때문에 기본적인 remove를 하였을 때처럼 파일목록을 출력하지 않고 해당 파일의 모든 백업 파일을 삭제합니다.

recover : backup파일을 통해서 기존 파일을 복구합니다. 즉, /home/backup/12자리의 시각 디렉토리 밑의 파일을 통해 사용자 디렉토리 밑의 파일들에 복구를 진행합니다. 해당 파일이 삭제된 경우에는 파일을 원래 위치에 복구합니다. 현재 존재하는 파일과 내용이 다르지 않은 경우 내용이 변하지 않았음을 출력하고 복구를 진행하지 않습니다. 또한 복구를 진행한 backup디렉토리 밑의 파일은 삭제됩니다.

-d 옵션 : 파일의 위치를 입력받는 것이 아닌 디렉토리 경로를 입력받아서 해당 디렉토리 안에 있는 디렉토리를 제외한 모든 파일에 대하여 recover를 진행합니다.

-r 옵션 : 디렉토리 경로를 입력받아서 BFS를 재귀적으로 사용하여 디렉토리 내부의 모든 파일들과 하부 디렉토리의 파일들까지 recover를 진행합니다.

-l 옵션 : 기존의 recover는 복구할 파일을 선택할 수 있었던 반면 l옵션의 경우 가장 최근에 backup한 파일로 복구를 진행합니다.

help : 사용할 수 있는 전체 명령어에 대한 설명을 출력합니다. help 명령어 순으로 입력하여 원하는 명령어를 특정지어 출력할 수 있으며 그냥 help만을 입력할 경우 전체 명령어가 출력됩니다.

3.상세설계

<순서도>

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명

<함수 및 모듈 구성>

우선적으로 로그를 이용하여 대부분의 과정이 진행됩니다. 프로그램이 실행됨에 따라 Init함수를 실행시킵니다. Init함수의 경우, backup 디렉토리가 없는 경우에 생성해줍니다. 그리고 ssubak.loh파일이 없는 경우에도 생성해줍니다. ssubak.log 파일 안에 있는 내용들을 한 줄씩 읽어들입니다. 읽어드린 로그를 기준으로 “backuped”가 쓰였을 때는 아래의 backupNode에 backupPath와 originPath를 저장해서 링크드리스트 형태로 만듭니다. 위와 같은 과정을 위해서 strtok함수를 사용하여 찾습니다. “removed”와 “recovered”가 쓰인 경우에는 기존의 backupNode의 링크드리스트에서 해당 backupNode를 제거합니다. 이렇게 Init함수가 실행되고 나면 현재 사용자가 접근할 수 있는 backup파일들의 링크드리스트가 생기게 됩니다. 이를 이용하여 사용자는 backup 함수, reomove함수, recover함수를 각각 명령어를 통해 실행시킬 수 있습니다. main 함수에서는 getopt함수를 통해서 옵션을 입력받고 해당 옵션이 입력되었을 경우에 따라 케이스를 분류하여 시행되게 하였습니다. 잘못된 옵션이나 경로가 들어왔을 경우에는 에러메세지를 출력합니다.

아래는 Init과 옵션들을 사용하기 위한 header파일에 구현되어 있는 구조체와 함수들입니다.

//백업한 내용들을 담아두기 위한 구조체입니다.

typedef struct backupNode {

char backupPath[PATHMAX];

char newPath[PATHMAX];

char originPath[PATHMAX];

char \*directoryName;

struct stat statbuf;

int countNum;

struct backupNode \*next;

struct backupNode \*prev;

} backupNode;

//백업 노드의 시작점을 담아두기 위한 구조체입니다.

typedef struct fileList {

struct backupNode \*head;

}fileList;

//-r옵션을 위해서 파일들의 링크드리스트를 만들기 위해 사용하는 구조체입니다.

typedef struct fileNode {

char path[PATHMAX];

backupNode \*head;

struct fileNode \*next;

} fileNode;

//-r옵션을 위해서 dir경로들을 링크드리스트로 만들기 위해 사용하는 구조체입니다.

typedef struct dirNode {

char path[PATHMAX];

char backupPath[PATHMAX];

fileNode \*filehead;

struct dirNode \*next;

} dirNode;

//dir 경로들의 head를 저장하기 위한 구조체입니다.

typedef struct dirList {

struct dirNode \*head;

} dirList;

//backupNode를 추가하는 함수입니다.

backupNode\* addBackupNode(backupNode \*head,char originPath[], char backupPath[]) {

backupNode\* newNode = (backupNode\*)malloc(sizeof(backupNode));

backupNode\* curr = (backupNode\*)malloc(sizeof(backupNode));

struct stat tmpbuf;

if (newNode == NULL) {

fprintf(stderr, "Memory allocation failed\n");

exit(1);

}

strcpy(newNode->originPath, originPath);

strcpy(newNode->backupPath, backupPath);

newNode->next = NULL;

if(head==NULL){

backupNode\* newHead = (backupNode\*)malloc(sizeof(backupNode));

newHead = head;

newHead->next = newNode;

return newHead;

//헤더를 동적할당해두면 이게 헤더에 데이터가 쓰이지않음

}

else{

curr = head;

while(curr->next !=NULL){

curr = curr->next;

}

curr->next = newNode;

}

return head;

}

//backupNode를 삭제하는 함수입니다.

backupNode\* deleteBackupNode(backupNode \*head, char \*deletePath){

backupNode \*tempNode;

tempNode = head;

backupNode \*deleteNode;

if((tempNode->next->next ==NULL)&&(strcmp(deletePath,tempNode->next->backupPath)==0)){

//한 개만 남았을 때에 대한 예외를 처리해줘야한다.

head->next =NULL;

return head;

}

while(tempNode->next != NULL){

if(strcmp(deletePath,tempNode->next->backupPath)==0){

deleteNode=tempNode->next;

tempNode->next= tempNode->next->next;

deleteNode->next =NULL;

free(deleteNode);

return head;

}

tempNode=tempNode->next;

}

return head;

}

//remove, recover함수등에서 사용하기 위해 originPath를 입력하여 해당 originPath를 가족 있는 backupNode를 찾는 함수입니다.

backupNode\* searchNode(backupNode \*head, char searchPath[]){

backupNode \*tempNode;

tempNode = head;

int count =0;

backupNode \*newHead = (backupNode\*)malloc(sizeof(backupNode));

while(tempNode ->next !=NULL){

tempNode = tempNode -> next;

if(strcmp(tempNode -> originPath,searchPath)==0){

newHead = addBackupNode(newHead, tempNode->originPath,tempNode->backupPath);

}

}

return newHead;

}

//-r옵션을 사용하기 위해서 필요한 queue를 위한 구조체입니다.

typedef struct queue{

dirNode \*first;

dirNode \*last;

}queue;

//queue를 추가하고 삭제하는 함수입니다.

void enqueue(queue\* Queue, char\* dirPath){

dirNode\* newNode = (dirNode \*)malloc(sizeof(dirNode));

strcpy(newNode->path,dirPath);

newNode->next=NULL;

if(Queue->first == NULL){

Queue->first = newNode;

Queue->last = newNode;

return;

}

else{

dirNode \*tempNode;

tempNode = Queue->last->next;

Queue ->last->next = newNode;

newNode -> next = tempNode;

Queue -> last = newNode;

return;

}

}

dirNode\* dequeue(queue\* Queue){

dirNode \*tempNode;

tempNode = Queue->first;

Queue->first = Queue->first->next;

return tempNode;

}

//backupRecursive함수를 통해 재귀함수의 형식으로 BNF를 시행시킵니다. addFileNode는 해당 디렉토리 안의 파일들을 노드로 만들어서 링크드리스트로 만들고 addDirNode는 file의 링크드리스트와 현재 디렉토리의 경로를 저장한 dirNode의 링크드리스트를 생성합니다.

fileNode \*addFileNode(fileNode \*head,char \*path){

fileNode\* newNode = (fileNode\*)malloc(sizeof(fileNode));

strcpy(newNode->path,path);

newNode ->next = NULL;

if(head==NULL){

head=newNode;

}

else{

fileNode\* tempNode;

tempNode = head;

while(tempNode->next!=NULL){

tempNode = tempNode->next;

}

tempNode ->next = newNode;

}

return head;

}

dirNode \*addDirNode(dirNode\* head, fileNode\* headFile,char\* path){

dirNode\* newNode = (dirNode\*)malloc(sizeof(dirNode));

newNode ->filehead = headFile;

newNode ->next = NULL;

strcpy(newNode->path,path);

if(head == NULL){

head = newNode;

return head;

}else{

dirNode\* tempNode;

tempNode = head;

while(tempNode -> next !=NULL){

tempNode = tempNode->next;

}

tempNode->next = newNode;

return head;

}

}

//backuprecursive 함수는 BNF를 재귀적으로 사용하여 구하는 -r옵션을 위해 사용되는 함수입니다.

void backupRecursive(queue\* Queue,char \*path,int backupOption){//backup뿐만 아니라 다른 명령어의 -r옵션에 대하여 사용가능합니다.

struct stat statbuf,tempbuf;

struct dirent \*\*namelist;

char\* tempPath = (char \*)malloc(sizeof(char \*)\*PATHMAX);

fileNode \*headFile = (fileNode\*)malloc(sizeof(fileNode));

fileNode \*forCheckFile;

dirNode \*nextDir;

backupNode \*forDirBack = (backupNode \*)malloc(sizeof(backupNode));

int cnt;

if(lstat(path,&statbuf)<0){

fprintf(stderr,"ERROR: lstat cuase error\n");

exit(1);

}

if((cnt = scandir(path,&namelist,NULL,alphasort)) == -1){

fprintf(stderr,"ERROR: cannot scan dir.\n");

exit(1);

}

for(int i = 0; i <cnt;i++){

if(!strcmp(namelist[i]->d\_name, ".") || !strcmp(namelist[i]->d\_name, "..")) continue;

strcpy(tempPath,path);

strcat(tempPath,"/");

strcat(tempPath,namelist[i]->d\_name);

if(lstat(tempPath,&tempbuf)<0){

fprintf(stderr,"lstat cause error.\n");

exit(1);

}

if(S\_ISDIR(tempbuf.st\_mode)){

enqueue(Queue,tempPath);

} //디렉토리인 경우에는 queue에 추가해줍니다.

if(S\_ISREG(tempbuf.st\_mode)){

headFile = addFileNode(headFile,tempPath);

} //파일인 경우에는 파일 링크드리스트에 추가해줍니다.

}

int count =0;

if(backupOption!=0){

forCheckFile = headFile;

forDirBack = searchDir(forFileList->head,path);

while(forDirBack->next!=NULL){

forDirBack= forDirBack->next;

while(forCheckFile->next!=NULL){

forCheckFile=forCheckFile->next;

if(strcmp(forCheckFile->path,forDirBack->originPath)!=0&&(count =0)){

headFile = addFileNode(headFile,forDirBack->originPath);

count ++;

}

}

count = 0;

}

}

mainDirList-> head = addDirNode(mainDirList->head,headFile,path);

if(Queue->first ==NULL){

return;

}

nextDir=dequeue(Queue);

backupRecursive(Queue,nextDir->path,backupOption);

}

//날짜 정보를 가져올 수 있는 getDate 함수와 해싱하여 해싱값을 비교하여 파일이 변경되었는지 확인할 수 있는 md5함수입니다.

char \*getDate() {

char \*date = (char \*)malloc(sizeof(char) \* 14);

time\_t timer;

struct tm \*t;

timer = time(NULL);

t = localtime(&timer);

sprintf(date, "%02d%02d%02d%02d%02d%02d",t->tm\_year %100, t->tm\_mon + 1, t->tm\_mday, t->tm\_hour, t->tm\_min, t->tm\_sec);

return date;

}

int md5(char \*target\_path, char \*hash\_result)

{

FILE \*fp;

unsigned char hash[MD5\_DIGEST\_LENGTH];

unsigned char buffer[SHRT\_MAX];

int bytes = 0;

MD5\_CTX md5;

if ((fp = fopen(target\_path, "rb")) == NULL){

printf("ERROR: fopen error for %s\n", target\_path);

return 1;

}

MD5\_Init(&md5);

while ((bytes = fread(buffer, 1, SHRT\_MAX, fp)) != 0)

MD5\_Update(&md5, buffer, bytes);

MD5\_Final(hash, &md5);

for (int i = 0; i < MD5\_DIGEST\_LENGTH; i++)

sprintf(hash\_result + (i \* 2), "%02x", hash[i]);

hash\_result[HASH\_MD5-1] = 0;

fclose(fp);

return 0;

}

//해당하는 경로의 디렉토리를 접근하고 생성할 수 있을 때까지 반복하는 재귀함수이다.

char \*accessDirCheck(char \*newPath){

int idx;

char \*pastPath = (char \*)malloc(strlen(newPath) + 1);

char \*tempPath = (char \*)malloc(strlen(newPath)+1);

strcpy(tempPath,newPath);

for(idx = strlen(tempPath)-1; tempPath[idx] != '/'|idx==0; idx--);

strcpy(pastPath, tempPath);

pastPath[idx] = '\0';

if(access(pastPath,F\_OK)){

if(mkdir(pastPath,0777)<0){

free(tempPath);

return accessDirCheck(pastPath);

}

}

}

헤더에 담아둔 위의 함수들과 구조체를 통해서 ssu\_backup.c파일에서 기능에서 설명했던 부분들을 수행할 수 있습니다.

help함수의 경우에는 ssu\_help.c로 따로 정의하여 Makefile에서 컴파일 후 링킹 시켜주고 있습니다. 각각의 기능들에 대한 설명을 출력하는 부분들을 개별적인 함수로 만들어서 해당 기능에 대한 설명이 필요할 때 해당 함수를 수행하게끔 작성하였습니다.

void printUsage(void){

printf("Usage:\n");

}

void printBackup(void){

printf(" > backup <PATH> [OPTION]... : backup file if <PATH> is file\n");

printf(" -d : backup files in directory if <PATH> is directory\n");

printf(" -r : backup files in directory recursive if <PATH> is directory\n");

printf(" -y : backup file although already backuped\n");

}

void printRemove(void){

printf(" > remove <PATH> [OPTION]... : remove backuped file if <PATH> is file\n");

printf(" -d : remove backuped files in directory if <PATH> is directory\n");

printf(" -r : remove backuped files in directory recursive if <PATH> is directory\n");

printf(" -a : remove all backuped files\n");

}

void printRecover(void){

printf(" > recover <PATH> [OPTION]... : recover backuped file if <PATH> is file\n");

printf(" -d : recover backuped files in directory if <PATH> is directory\n");

printf(" -r : recover backuped files in directory recursive if <PATH> is directory\n");

printf(" -l : recover lates backuped file\n");

printf(" -n <NEW\_PATH> : recover backuped file with new path\n");

}

void printList(void){

printf(" > list [PATH] : show backup list by directory structure is file directory is file directory \n");

printf(" >> rm <INDEX> [OPTION]... : remove backuped files of [INDEX] with [OPTION]\n");

printf(" >> rc <INDEX> [OPTION]... : recover backuped files of [INDEX] with [OPTION]\n");

printf(" >> vi(m) <INDEX> : edit original file of [INDEX]\n");

printf(" >> exit : exit program\n");

}

void printHelp(void){

printf(" > help [COMMAND] : show commands for program\n");

}

backup, remove, recover은 아래 함수 프로토타입을 통해 설명하겠습니다.

<함수 프로토타입>

int BackupFile(char \*path, int option)

이는 파일을 백업하는 함수로 처음에 pathmax보다 경우에 에러를 출력하며 프로그램을 종료시킵니다. 그 후 경로를 realpath함수를 이용하여 절대경로를 얻습니다. 이렇게 얻은 경로 안에 /home/backup이 포함되거나 /home이라는 문자열이 포함되어있지 않는 경우에는 에러메세지를 출력하고 프로그램을 종료합니다. 그 후 filename을 구합니다. lstat함수를 이용하여 stat구조체를 리턴받고 directory인지 확인합니다. 전역변수로 선언하여 Init함수에서 링크드리스트를 저장해둔 forFileList를 통해서 앞서 로그를 이용해 만들었던 링크드리스트를 이동시키면서 같은 파일에 대한 백업파일이 존재하는지 확인합니다. 이때 매개변수로 받았던 option을 이용해서 -y옵션이 없는 경우 option의 값은 0이기에 같은 백업파일이 있는 경우에 메세지를 출력하고 프로그램을 종료합니다. 백업파일이 이미 존재하지 않는 경우에는 날짜 정보를 얻어 backup디렉토리에 접근한 후 새로운 path를 만들어서 그 path에 해당하는 디렉토리를 생성한 후에 내용을 백업해줍니다. 그 후 ssubak.log를 열어 해당내용을 기록합니다.

int BackupDir(char \* path,int option)

디렉토리에 대해서 앞선 함수인 BackupFile과 비슷한 절차를 진행합니다. 앞선 확인 절차를 동일하게 진행한 후 backupDir함수의 경우에는 파일이 경로로 들어오면 에러메세지를 출력하고 프로그램을 종료합니다. scandir함수를 사용하여 경로의 디렉토리 안의 파일들에 대한 백업을 진행합니다. 이때도 option을 통해서 강제로 백업파일을 만들지 결정합니다. switching이라는 변수를 사용하여 option이 0이고 이미 백업파일이 있는 경우에 백업을 진행하지 않도록 합니다.

int BackupForR(char\* path,int option)

-r옵션을 위해 만들어진 함수입니다. mainDirList가 헤더에 전역변수로 선언되어있기에 이에 대한 동적할당을 진행해줍니다. queue를 하나 생성하여 동적할당을 해주고 초기화시켜줍니다. 앞선 함수들과 동일한 확인 절차를 거친 후에 backupRecursive함수를 수행합니다. 이렇게 얻은 dirNode들을 순차적으로 수행시키면서 파일에 대한 백업을 전부 진행한 후에 서브 디렉토리에 들어가서 여기 안에 있는 파일들에 대한 백업을 진행해줍니다. 이를 통해서 BFS를 이용하여 디렉토리 구조 전부를 백업할 수 있습니다.

int BackupfileForR(char \*path, int option,char\* dateData,char\* dirData)

-r옵션을 위해 만들어진 backupfile함수로서 파일들의 디렉토리 상하 구조를 유지하기 위해서 매개변수로 dirData를 받고 또한 backup되는 시간도 같아야하기 때문에 dateData를 매개변수로 받습니다. 앞선 함수들과 같은 확인 절차를 거치고 /home/backup/12자리 시각의 하위 디렉토리의 위치만 달라질 뿐 Backupfile과 동작과정이 동일합니다.

int RemoveFile(char \*path)

백업파일을 삭제하는 함수입니다. path의 최대 길이를 초과하는지 확인하고 초과하는 경우 에러메세지를 출력하고 프로그램을 종료시킵니다. 삭제된 파일의 백업파일에 대한 삭제를 진행할 수 있기 때문에 realpath함수가 NULL을 리턴하는 경우에는 상대경로일때는 .을 제외하고 한 비트 이동시키고, 절대 경로의 경우에는 그대로 그 경로를 이용합니다. 위의 searchNode함수를 사용하여 originPath가 같은 파일들을 추려내어 forRemoveList라는 전역변수인 포인터의 헤더에 리턴받은 헤더를 저장시킵니다. 또한 이때 head->next ==NULL이면 백업해둔 파일이 없기 때문에 lstat의 결과가 -1이면 존재하지 않는 경로임을 에러메세지로 출력하고, 디렉토리 경로인 경우인지도 확인하여 에러메세지를 출력합니다. tempNode -> next가 NULL인 경우에는 삭제할 파일이 없어서 에러메세지를 출력하고 프로그램을 종료합니다. tempNode->next->next가 NULL인 경우에는 삭제할 파일이 하나이기 때문에 파일을 삭제하고 파일이 한 개만 존재하는 디렉토리의 경우 파일이 삭제된 후에 디렉토리가 비게 되면 디렉토리 또한 삭제해줍니다. 위의 과정이 끝난 후에는 ssubak.log를 열어서 로그를 작성시켜줍니다. 또한 파일이 여러 개인 경우에는 backup된지 오래된 순서부터 인덱스를 매겨 출력하여 보여줍니다. 사용자가 0을 입력하고 아무런 결과없이 프로그램을 종료하며 사용자가 인덱스를 선택하여 누르게 되면 해당 인덱스의 백업파일을 삭제합니다. 이 과정도 도일하게 과정이 완료된 후에는 로그를 작성합니다.

int RemoveForA(char \*path )

-a옵션을 처리하기 위한 함수입니다. 앞의 대부분의 과정들은 RemoveFile함수과 같습니다. 다만 백업파일의 목록을 출력하지 않고 모든 백업파일에 대해 삭제하는데서 차이점을 가집니다.

int RemoveDir(char \*path,int option)

-d옵션을 위해 사용되며 확인하는 절차는 위와 동일하게 진행합니다. 이때도 scandir로 입력받은 path의 파일들을 확인하고 option의 값이 0 인 경우에는 -a옵션을 적용하지 않고 option이 1인 경우에는 -a옵션을 적용하여 각각 RemoveFile과 RemoveForA함수를 호출합니다.

int RemoveForR(char \*path, int option)

-r옵션을 위해 사용되며 앞선 경로의 길이와 올바른 경로인지에 대한 확인 절차를 수행합니다. 그 후에는 backupRecursive함수를 수행하여 초기화해둔 queue를 집어넣고 전역변수로 선언해 둔 mainDirList를 사용하여 BNF구조로 remove될 수 있게끔 해줍니다. 옵션에 따라 RemoveFile 과 RemoveForA함수를 다르게 사용해줍니다.

int RecoverFile(char \*path,int option)

recover명령어를 수행하기 위한 함수입니다. pathmax를 초과하여 들어오는 경우에 에러메세지를 출력하며 프로그램을 종료시킵니다. realpath함수를 사용하였을 때 리턴 값이 NULL인 경우에는 .을 기준으로 절대 경로와 상대경로를 나눠서 originPath를 만들어줍니다. 그렇게 만들어준 originPath를 이용하여 searchNode함수를 수행시켜줍니다. head->next가 NULL인 경우에는 원래 경로상에 존재하는지 확인하고 존재하는 경우에는 디렉토리 경로인지 확인합니다. 둘 중 하나라도 아닌 경우에는 에러메세지를 출력하고 프로그램을 종료합니다. filename을 구해서 저장시켜줍니다.

해당 파일이 존재하는 디렉토리가 없는 경우에는 생성해줍니다. 또한 option이 1인 경우에는 -l옵션을 사용한 것으로 간주해 tempNode를 옆 노드로 이동시켜 최신 노드로 불러옵니다. 그 후에는 옆의 노드가 하나인 경우에는 값이 변했는지 md5를 통해 확인하고 같으면 같다는 메세지와 함께 프로그램을 종료합니다. 다른 경우에는 백업되어있는 내용을 기존 경로의 파일에 덮어씌우고 로그를 작성해줍니다. 백업파일이 여러개인 경우에는 목록을 띄우고 그 중에서 선택하여 복구할 수 있도록 합니다.

int RecoverDir(char \*path,int option)

-d옵션을 수행하기 위한 함수로서 초반의 검증과정은 위와 동일하게 진행됩니다. scandir함수를 통해서 함수와 디렉토리를 구분하고 디렉토리를 제외한 함수에 대해서 recover함수를 수행합니다. 이때 옵션값에 따라 RecoverFile에 대한 옵션매개변수를 다르게 넣어줍니다.

int RecoverForR(char \*path, int option)

-r옵션을 위한 함수입니다. 앞선 확인 과정들은 동일하게 진행되며 queue를 생성하여 초기화 시켜줍니다. 그 후에 backupRecursive함수를 이용하여 전역변수 포인터인 mainDirList를 사용하며 BFS를 사용하여 파일에 대한 recover을 먼저하고 그 후에 서브 디렉토리 안의 파일들을 확인하여 recover를 진행시켜줍니다. 이때도 입력받은 option매개변수를 통해 -l옵션을 수행하는 것인지 아닌지를 구분할 수 있습니다.

4.실행결과

help 명령어의 실행결과 입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

인자를 입력하지 않았을 경우

텍스트, 스크린샷, 폰트이(가) 표시된 사진

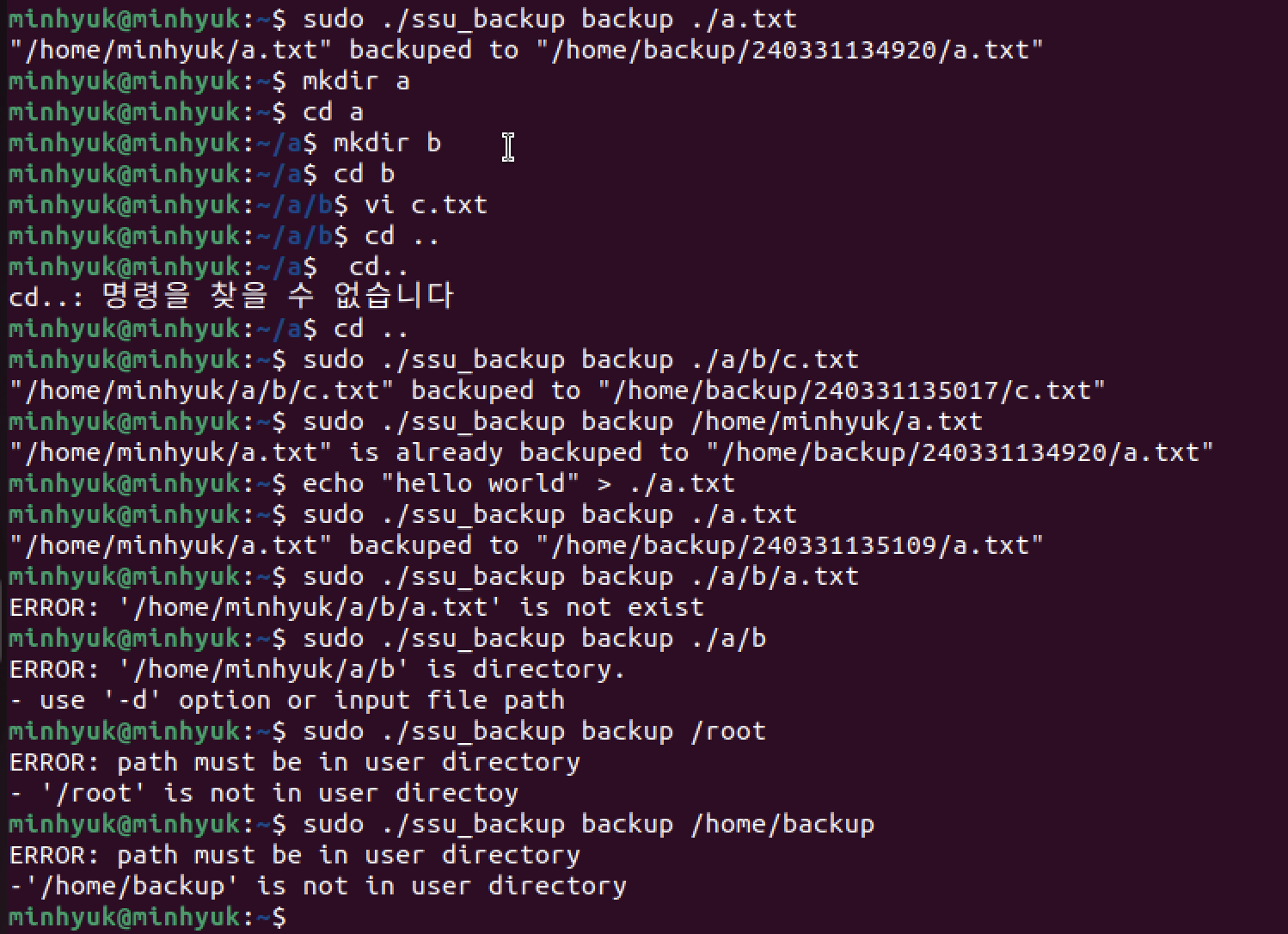
자동 생성된 설명

잘못된 명령어 입력 시

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

기본적인 backup명령어를 사용하였을 때 발생할 수 있는 에러들과 정상적으로 작동되었을 때의 결과입니다.



-d옵션을 적용하여 backup을 하였을 때의 결과입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-r옵션을 적용하여 backup을 하였을 때의 결과입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

-y옵션을 적용하여 backup을 하였을 때의 결과입니다.텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

아래는 remove명령어를 수행했을 결과입니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

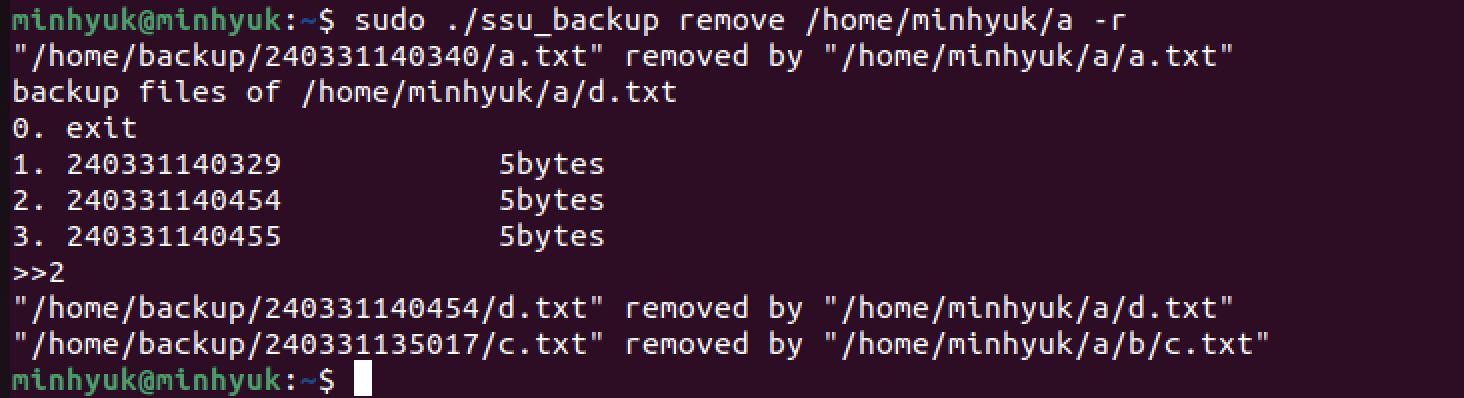
자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

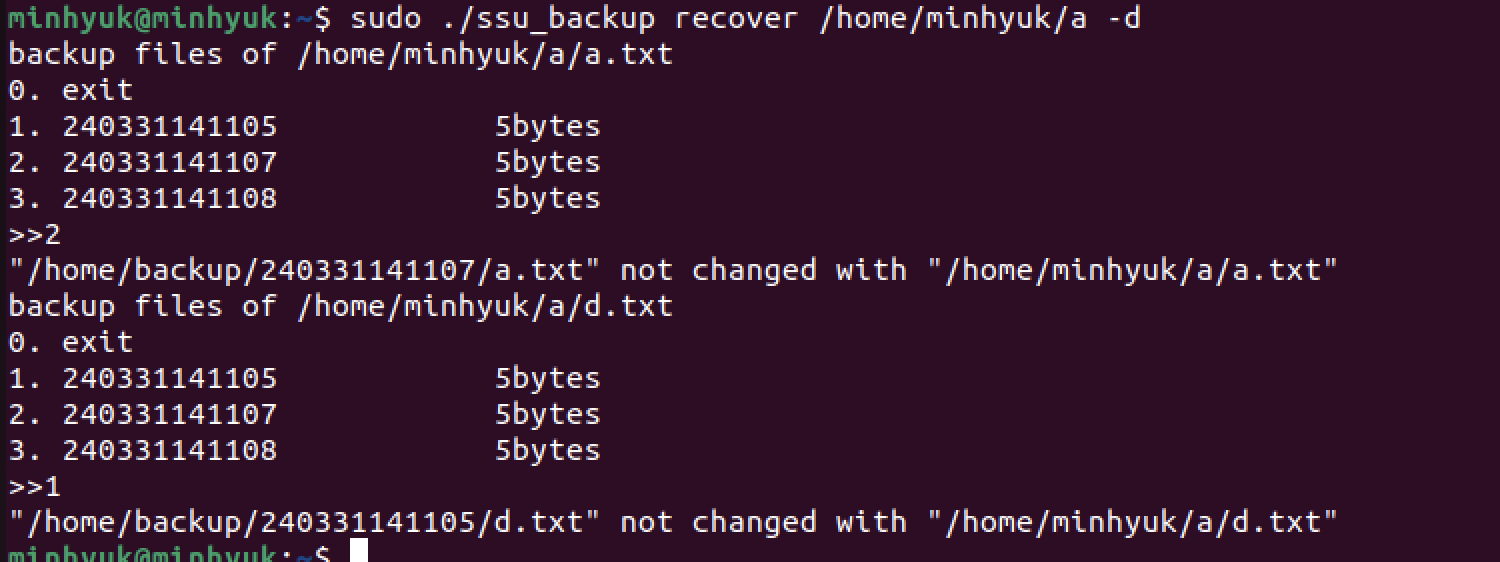
자동 생성된 설명



아래는 recover명령어를 수행했을 때의 결과입니다.텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

-d옵션을 수행했을 때의 결과입니다.



-r옵션을 수행했을 경우입니다. 밑은 -l옵션을 사용하였을 경우입니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명