학과 : 컴퓨터학부 학번 : 20192392 이름 : 김한승

1. 과제 개요

ssu\_backup 백업 프로그램입니다. 기존 리눅스 명령어로 사용할 수 있는 ls, vi(m) 내부 명령어를 사용할 수 있으며, 기존에 정의되지 않은 명령어 나 명령어를 잘못 입력하였을 경우 “help” 내부 명령어를 입력하였을 경우에 사용자가 명령어를 확인할 수 있도록 usage를 출력합니다. 동일한 파일의 검사를 위해 해시함수 <md5 | sha1> 을 지원합니다. 모든 내부 명령어는 fork() / exec()류 함수를 통해 자식 프로세스를 불러내어, 명령어를 수행합니다. 자식 프로세스를 통해 내부 명령어 수행 후, 다시 ssu\_backup 프로그램으로 돌아오는 것을 반복하는 것입니다. exit 명령어를 통해 백업 프로그램 종료.

프로그램 실행 시, 사용자는 ssu\_backup 프로그램 실행 인자로 사용할 해시함수를 입력합니다. <md5 | sha1>

처음 백업 프로그램을 실행하면, 홈 디렉토리 하위 backup 디렉토리를 생성합니다. 백업 프로그램이 백업된 파일과 디렉토리를 관리할 디렉토리를 생성하는 것입니다.

프로그램 만에 특징으로 3가지 대표 내부 명령어가 있습니다. add / remove / recover 로 3가지입니다.

1. add 명령어의 경우, <$HOME ,~> 경로 위에 백업 하기를 원하는 일반 파일이나 디렉토리를 백업 할 수 있습니다.

이때 각, 백업 파일 명 형식은 “백업하는원본파일명\_YYMMDDHHMMSS” 형태입니다.

백업 파일은 <$HOME ,~> 경로의 하위 backup 디렉토리로 백업 되며, add 명령어로 백업 할 때 해당 디렉토리의 경로를 포함하거나, 디렉토리 내 파일이나 서브 디렉토리를 포함 시, 백업 하지 않습니다.</홈디렉토리/backup 경로 포함>

또한, <$HOME ,~> 경로 위에 있지 않을 경우에도 백업 하지 않습니다.

동일한 백업 파일이 존재하는지 사용자가 처음 실행 시 입력한 해시함수를 통해 원본 파일과 존재하는 백업 파일을 비교하며, 이때 동일한 해시 값을 가진 백업 파일이 존재할 경우, 백업을 진행하지 않습니다.

사용자가 수정을 한 후, add 명령어를 통해 백업 할 시, 기존 백업 파일과 다른 원본 파일을 백업 하려고 시도하는 것이므로, 백업이 진행되며, 이는 같은 이름의 여러 다른 백업을 가질 수 있음을 의미합니다.(같은 이름의 백업파일이지만, 다른 내용을 가지고 있으며, 다른 백업 시각을 갖습니다 ex ) <파일명\_서로 다른 백업시각.>

디렉토리를 백업 하는 경우에는 입력된 경로의 디렉토리로부터 하위에 모든 일반 파일 및 디렉토리를 백업합니다.

이때, 백업 할 파일이 없는 디렉토리의 경우는 디렉토리를 생성하지 않습니다.

2. Remove 명령어의 경우, 홈 디렉토리 하위 /backup에 백업 되 있는 일반 파일 및 디렉토리를 지울 수 있습니다. 원본 파일이 존재하더라도, 백업 파일이 존재하지 않거나, 원본 파일과 백업 파일 둘 다 존재 하지 않을 경우 remove 명령어는 usage를 출력 후, 명령어 수행을 끝냅니다.

remove의 경우에는 경로상 원본파일이 없을 경우에도, 원본이 백업이 되 있다면 /backup 하위 백업 파일을 지울 수 있습니다. 원본을 수정하면서 여러 같은 이름의 다른 백업 파일을 생성해 낼 수 있었는데, remove 명령어는 같은 이름의 다른 백업 파일들 중, 어떤 백업 파일을 지울지 고를 수 있도록 지원합니다.

‘-a’ 옵션을 통해 디렉토리를 지울 경우, 재귀 적으로 해당 디렉토리와 모든 파일 및 서브 디렉토리를 삭제 할 수 있으며, ‘-a’ 을 받은 파일을 지울 경우, 존재하는 해당 원본 파일에 대한 모든 백업 파일을 지웁니다.

ex) 백업파일1\_시각1 , 백업파일1\_시각2 , 백업파일1\_시각3 -a 옵션을 받고 명령어를 수행 시, 어떤 파일을 지울지 고르지 않고 3개 파일 모두를 지웁니다.

‘-c’ 옵션을 받을 경우 <FILENAME>을 인자로 받지 않습니다. remove -c 의 입력에만 동작하며, 이 경우, 홈디렉토리 하위 /backup 에 존재 하는 모든 일반 파일 및 디렉토리를 삭제합니다. 삭제가 끝난 후, 결과로 몇 개의 파일과 디렉토리를 지웠는지 출력하며, 아무 파일이나 디렉토리도 지우지 못했을 경우에는, “no file(s) in the backup”을 출력하고 명령어 수행을 종료합니다,

3. Recover 명령어의 경우, 홈 디렉토리 하위 /backup에 백업 되 있는 일반 파일 및 디렉토리를 복원할 수 있습니다.

recover 명령어와 동일하게 원본 파일이 없을 경우에도, 원본이 백업이 되 있다면 /backup 하위 백업 파일을 복원할 수였습니다.

recover 명령어 또한 add, remove 명령어와 동일하게 ‘-d’ 옵션을 받으면, 재귀적으로 해당하는 파일의 백업파일을 복원합니다. 이때, 백업파일이 여러 개 존재할 경우마다 리스트로 어떤 백업파일을 복원할 것인지 고를 수 있습니다.

recover 명령어는 ‘-n <NEWNAME>’ 옵션을 제공합니다. 이는 백업할 때 사용한 경로 와 다른 경로로 해당 파일을 복원할 수도, 백업 할 때와 다른 파일명으로 복원할 수 있습니다.

recover 명령어는 복원을 시도하는 백업 파일과 복원하는 곳의 원본 파일이 같은 파일이라면 복원을 진행하지 않습니다.

recover 함수 또한, 사용자가 처음에 입력한 해시 함수를 통해 해시 값을 비교하여 파일이 동일한지 비교합니다.

2. 구현 기능.

0. Makefile : 링크드 리스트를 내부적으로 사용하는 add,remove,recover에 경우는 linked\_list.o 오브젝트 파일을 컴파일 시, 포함 시켰으며, ssu\_backup\_u.c에 세개의 유틸리티 함수를 사용하기 위해 ssu\_backup\_u.o 오브젝트 파일을 linked\_list.o 오브젝트 파일을 만들 때와 add,remove,recover 실행 파일을 만들 때 포함시켰습니다. 특이점으로, linked\_list.o 오브젝트 파파일 만들 때, sah1, md5 해시 값을 만들기 위한 openssl 관련 라이브러리 컴파일을 위해 -lcrypto -lssl 을 넣어주었고,

Md5 | sha1 함수가 3.0.0 이후 deprecated 되었음을 확인한후, -DOPENSSL\_API\_COMPAT =10101을 통해 라이브러리 API 레벨을 v1.1.1 로 컴파일 하며 옵션을 통해 낮추어 줬습니다.

1. header.h : 헤더 내에 과제 명세서에 적혀진 대로의 최대 파일 경로 길이와 최대 파일 이름 길이

홈 경로 , 작업 경로를 getenv() 로 받아서 헤더 가드(HEADER\_H) 로 선언했습니다.

2. ssu\_backup\_u.c , ssu\_backup\_u.h : ssu\_backup 프로그램 전반적으로 사용할 유틸리티 함수들을 선언했습니다.

- 1. void getBackupPath(char \*absolute\_path, char \*backup\_path, char\* PATH\_BACKUP\_DIRECTORY);

- 2. void getBackupDirectoryPath(char \*PATH\_BACKUP\_DIREOCTRY);

- 3. char\* getProcessPath(char \*path);

getBackupPath() 함수는 절대경로와 백업 경로를 저장할 백업 경로의 포인터, 현재 백업 디렉토리 경로의 위치를 받아와서, backup\_path에 absolute\_path가 백업이 된다면, /backup 디렉토리 하위의 어느 경로로 들어가야 되는지,

backup\_path에 경로를 받을 수 있는 함수입니다.

getBackupDirectoryPath() 는 PATH\_BACKUP\_DIRECTORY 포인터에 /backup 디렉토리 경로를 받는 함수입니다.

사용자가 프로그램을 실행하는 위치에 따라서, 저의 홈 디렉토리는 “/home/hanseung” 이지만, 사용자마다 다른 경로를 가지기 때문에, header.h 에서 받은 getenv(“HOME”), “/backup” 두 문자열을 합쳐서 파라미터로 받은 PATH\_BAKCUP\_DIRECTORY에 받습니다.

getProcessPath()는 realpath() 사용함에 있어서 링크 파일등에 처리에 문제가 있어서 만든 함수입니다. 링크 파일에 경우는 realpath()에 넣을 시, 링크파일이 가리키는 파일에 대한 절대 경로를 반환합니다. 이에 따라, 현재 백업 하거나 복원하려는 파일이 제대로 일반 파일인지, 디렉토리인지 확인을 할 수 없었기 때문에, 직접 ~와 . 와 .. 등을 처리하여서 절대경로를 만들고 함수가 직접 반환하는 함수입니다.

3. ssu\_backup.c : 메인이 되는 프로그램 c 코드입니다. 해당 c 파일에서는 사용자가 프로그램을 시작하기전 사용할 해시 함수를 인자로 받으며 시작을 하고, “20192392>”를 출력하며, 내부 명령어들의 실행을 받을 입력 대기 상태가 됩니다.

각 내부 명령어를 받은 후, (exit 명령어를 제외한) 모든 내부 명령어는 fork() execv()를 활용하여, 내부 명령어를 수행합니다.

4. help.c : printf()의 연속입니다. Help 명령어나 정의 되지 않은 명령어(엔터 ‘’\n 제외)를 입력 받았을 경우. Usage 출력을 위해 만들어졌습니다.

5. add.c : ssu\_backup 프로그램에서 내부 명령어 add 명령어를 수행합니다. 먼저 첫번째 인자로 들어온 경로를 상대 경로인지, 절대경로인지 에 따라 다르게 처리합니다. 상대 경로일 경우 현재 디렉토리 경로를 붙여, ssu\_backup\_u.c에 정의된 getProcessPath()를 통해 절대 경로로 가공을 한 후, 파일 정보를 받으며, 그후, 예외 처리 접근권한, 일반파일인지, 디렉토리인지, 디렉토리일경우 d 플래그를 받았는지, ~ 디렉토리를 벗어났는지, 백업 디렉토리를 포함 안 하였는지, 예외 처리를 거쳐, 디렉토리일경우 linked\_list.c에 선언된, backupFiles()을 호출합니다. 파일일 경우, backupFile()을 호출합니다. 위에 예외 처리 및 경로의 가공은 remove 와 recover 명령어 수행에서도 동일합니다.

6. remove.c : ssu\_backup 프로그램에서 내부 명령어 remove 명령어를 수행합니다. 경로를 가공하고, 예외를 처리하며, 인자를 getopt()를 통해 처리하고, remove에 경우에는 -c 옵션을 받을 경우 다른 인자를 받으면 안되며, remove -c 를 통해 모든 백업 디렉토리 하위 파일 및 서브 디렉토리를 지우며, 이때 linked\_list.c 에 선언된, removeAll()를 호출합니다. Remove 와 recover 가 add와 가장 다른 점은, add에 경우는 원본 파일을 백업하는 것이므로, 원본이 무조건 존재하나, remove 와 recover에 경우는 원본 파일이 없을 수도 있다는 점입니다. 이로 인하여, 파일이 존재하는지 여부는 remove/recover 에서는 경로에 해당하는 백업 파일에 대해서 이루어집니다. Opendir()을 통해 dirp1가 바로 열린다면, 백업 경로내에 백업 디렉토리가 열릴 것이며, dirp2가 열린다는 것은 백업 경로내에 백업파일 바로 위 디렉토리를 오픈 한 것입니다.

Ex) /home/hanseung/backup/test/test.c\_YYMMDDHHMMSS -> 파일에 경우는 \_YYMMDDHHMMSS 라는 timeString이 붙어있으므로, 상단에 test 디렉토리의 경로를 오픈하려고 시도하는 것.

Ex) /home/hanseung/backup/test 에 경우 -> opendir로 먼저 열려고 시도 시, 존재한다면 파일에 경우와는 다르게 timeString이 붙어있지 않으므로, 바로 열리는 것.

Opendir() 백업 경로를 바로 열었을 때 열린다. -> 디렉토리를 찾으려 한다.

Opendir() 백업 경로에서 /파일명을 제거 후, 상위 디렉토리를 열린다 -> 파일을 찾으려 한다. -> 파일을 찾는다면 제거 및 복원 실시.

Opendir() 두 경우 모두 열리지 않을 시는 , 백업된 디렉토리나 파일이 존재하지 않는다.

Recover 에서도 동일한 방법으로 백업된 파일과 디렉토리를 찾으며, 이를 통해 원본 파일이 존재 하지 않더라도 원본의 경로를 통해 백업된 백업파일이나 디렉토리를 접근할 수 있었습니다.

Remove -c 명령어를 통한 처리는 removeAll()을 호출하고,

그 외, opendir()로 dirp1이 NULL이 아닌 경우 디렉토리를 삭제하려고 시도하는 것이므로, removeFiles()를 호출합니다.

이때 인자로는 디렉토리 링크드 리스트를 만들어 removeFiles()에 인자로 넘겨줍니다.

Dirp1이 NULL이며, dirp2가 NULL이 아닐 시, 삭제할 파일을 일단 찾습니다. 백업 경로에 상위 디렉토리를 연 것 이므로, 입력한 인자 <FILENAME> 에 대한 삭제할 파일이 존재한다면, 파일 링크드 리스트를 만들어 removeFile()를 호출하며 인자로 넘겨줍니다.

7. recover.c : ssu\_backup 프로그램에서 내부 명령어 recover 명령어를 수행합니다. 위 remove와 동일하게 원본 파일이 없는 경우에도 백업 파일이 존재하면 복원을 실시해야 하므로, dirp1 : opendir() 백업 경로로 시도, dirp2 : opendir() 백업 경로에서 마지막 ‘/’ 앞 문자열을 통해 백업 경로 상위 디렉토리를 열려고 시도 하여, dirp1 이 NULL 이 아니면 디렉토리를 복원하고, dirp1이 NULL이며, dirp2가 NULL이 아니라면, 해당하는 <FILENAME> 에 대한 백업 파일을 찾아 복원을 시도합니다. Add와 remove때와 마찬가지로, 예외 처리와 경로를 절대 경로로 가공하고, relapath()를 사용하는 것 또한 동일합니다.예외 처리를 모두 거친 후, 디렉토리에 대한 복원인지와 파일에 대한 복원인지 여부를 dirp1 과 dirp2의 값을 통해 결정한후,디렉토리 복원에 경우는 dirp1을 열 때 사용한 경로와 디렉토리 포인터, 디렉토리 링크드 리스트를 인자로 넘겨,

recoverFiles()를 호출합니다.파일에 대한 복원에 경우는 dirp2를 열 때 사용한 경로에서 <FILENAME>과 일치하는 백업 파일을 찾은 후, 파일 노드를 만들어 파일 링크드 리스트를 만든 후, reocverFile()에게 인자로 넘겨, 호출합니다.

8. linked\_list.c / linked\_list.h : 내부 명령어내에서 링크드 리스트를 통해 파일을 백업/제거/복원을 위해 만들었습니다.

내부 명령어 add.c, remove.c, recover.c에서 아래 정의된 함수들을 호출하며, 디렉토리 노드 node 와 파일 노드 node\_f를 정의하고, 이 두 노드에 대한 링크드 리스트를 만들며, /backup 디렉토리 하위에 원본을 백업, /backup 디렉토리 하위 파일을 삭제, 백업 파일들을 경로를 주어 복원하는 모든 역할을 합니다.

// directory node

- void push\_front(Node \*\*head, DIR \*dirp, char \*path);

- void push\_back(Node \*\*head, DIR \*dirp, char \*path);

- void pop\_front(Node \*\*head);

- void pop\_back(Node \*\*head);

- void insertNode(Node \*prevNode, DIR \*dirp, char \*path);

// file node

- void push\_back\_f(Node\_f \*\*head\_f, struct stat statbuf, char path[]);

- void pop\_front\_f(Node\_f \*\*head\_f);

// add

- void backupFiles(Node \*\*head, int flag\_md5, int flag\_sha1); // backup files(BFS)

- void backupFile(char \*filename, int flag\_md5, int flag\_sha1); // backup file

// remove

- void removeAll(int flag\_c); // remove /backup directory

- void removeFiles(Node \*\*head, int flag\_c, int \*deleteF\_count, int \*deleteD\_count); // directory remove(DFS)

- void removeFile(Node\_f \*\*head, int flag\_c, int count); // file remove

// recover

- void recoverFiles(Node \*\*head, int flag\_md5, int flag\_sha1, int flag\_n, char\* filename\_new); // directory recover(DFS)

- void recoverFile(Node\_f \*\*head\_f, int flag\_md5, int flag\_sha1, char\* filename\_new); // file recover

- void compareAndCopyFile(char\* source\_path, char\* dest\_path, int flag\_md5, int flag\_sha1);

// hash function

- void getMd5Sum(char \*filename, unsigned char\* md5Hash);

- void getSha1Sum(char \*filename, unsigned char\* sha1Hash);

// directory node

1. push\_front() : 새로운 노드를 생성한후 링크드 리스트의 맨 앞에 삽입.

- head는 이중 포인터로, 링크드 리스트의 헤드 노드의 주소를 가리킵니다. dirp는 디렉토리 포인터입니다.

path는 디렉토리의 경로입니다.노드를 동적 할당한후(malloc()) 노드에 dirp와 path 정보를 복사하여 노드에

저장 후, 새로운 노드를 헤드로 지정합니다. 그전에 헤드였던 노드를 새로운 노드 다음에 연결합니다.

2. push\_back() : 새로운 노드를 생성한 후 링크드 리스트의 맨 뒤에 삽입.

- 인자는 push\_front()와 동일합니다.노드를 동적 할당한후(malloc()) 노드에 dirp와 path 정보를 복사하여 노드에

저장 후, 링크드 리스트의 맨 마지막 노드에 새로운 노드를 연결합니다.

3. pop\_front() : 링크드 리스트의 맨 앞 노드를 삭제합니다.

- 현재 헤드가 가리키는 노드가 가리키는 다음 번째 노드를 헤드로 옮기고, 전에 헤드였던 노드는 삭제합니다.

4. pop\_back() : 링크드 리스트의 맨 뒤 노드를 삭제합니다.

- 링크드 리스트의 맨 마지막 노드를 삭제합니다.

5. insertNode() 인자로 dirp와 path를 받고, prevNode를 가리키는 포인터를 받습니다.

prevNode가 가리키는 다음 번 노드를 동적 할당 받은 새로운 노드가 가리키게 하고, 새로운 노드를 prevNode가 가리키게 합니다.

// file node

1. push\_front\_f() : 새로운 노드를 생성한후 링크드 리스트의 맨 앞에 삽입

- head 이중 포인터와 stat 구조체 정보, 파일의 경로를 인자로 받습니다. 동적으로 할당된 노드에 파일 구조체와 파일 경로를 복사한후, 파일 링크드 리스트 맨 앞에 삽입합니다. 그후, 헤드를 새로운 노드를 가리키게 합니다.

2. pop\_front()\_f : 파일 링크드 리스트의 맨 앞 노드를 삭제합니다.

- 파일 링크드 리스트의 맨 앞 노드를 삭제합니다.

// add

1. backupFiles() : 디렉토리를 재귀적으로 탐색한후, 백업 해야 할 파일이 존재하면, backupFile()를 호출하여 파일 을 백업합니다. 디렉토리 내 DIR 포인터를 통해 readdir() 하여서 받은 dirent 구조체에서 파일명이 “.” 과 “..” 이면 건너 뛰고, 나머지 파일들을 lstat()을 통해 열어 디렉토리라면 링크드 리스트에 다음으로 연결시켜, 현재 디렉토 리 하위에 파일과 디렉토리를 모두 탐색한후, 다음 하위 디렉토리를 탐색할 수 있도록 합니다.(BFS), 파일에 경우 는 backupFile()를 호출합니다. backupFile에 인자로는, 현재 파일 경로, flag\_md5, flag\_sha1 이 있습니다.

2. backupFile() : 인자로 받은 파일 경로를 통해 파일을 백업 시킵니다.

- 백업하기전 , 백업 경로를 얻습니다. Ssu\_backup\_u.c에 정의된, getBackupPath()

- 백업 경로상에 모든 상위 디렉토리가 없다면 만듭니다. Strtok()를 통해 /로 경로를 나누어 디렉토리를 만듭니다.

- Ex) /home/hanseung/backup/test/1.c : 1.c를 백업하기위해 test 디렉토리가 없다면 만듭니다.

- 백업 하려는 파일 상위에 디렉토리를 탐색하여서, 같은 이름의 같은 해시 값을 가진 백업 파일이 있는지 확인

- 원본과 완전 동일한 백업 파일이 존재한다면, 백업 하지 않고, 그것이 아니라면 백업을 진행합니다.

- 이때 백업을 진행할 때, fork() execvp()를 이용하여, cp 명령어를 파일의 경로와 파일의 백업 경로를 넘겨줍니다.

- 백업경로는 파일의 백업경로에 13개의 문자로 이루어진 백업 경로 + timeString(\_YYMMDDHHMMSS)

의 형태입니다,

- 파일의 중복 여부를 알기 위해 파일의 해시 값을 비교할 때, 인자로 받았던 flag\_md5 나 flag\_sha1 중 1로 플래 그 값이 지정 되 있는 해시 함수를 사용합니다. Linked\_list.c 에 정의된 getMd5Sum() | getSha1Sum() 사용.

// remove

1. removeAll() : remove -c 처리를 위한 함수. /’backup 디렉토리에 대한 링크드 리스트를 만든 후,

removeFiles()를 부름.

2. removeFiles() : 넘겨받은 디렉토리 노드들로 이루어진 링크드 리스트를 인자로 받은 후, 하위 디렉토리와 파일 들을 제거함.

3. removeFile() : 넘겨받은 파일 노드들로 이루어진 링크드 리스트를 인자로 받은 후, 링크드 리스트에 해당 파일 을 제거한다. 인자 중 count는 링크드 리스트에 count번에 해당하는 파일 노드 백업 파일을 제거합니다.

// recover

1. recoverFiles() : 디렉토리를 탐색하여 현재 복원할 파일과 일치하는 파일명을 가진 백업 파일

Add 하면서 다른 해시값을 가져서 백업이 여러 번 된 파일들을 묶어서 파일 링크드 리스트를 만든 후, recoverFile()를 호출합니다. 만약, 디렉토리 dirent 구조체 탐색 중 또 다른 디렉토리를 발견하면, recoverFiles()를 호출하기전 링크드 리스트 맨앞에 새로운 디렉토리 노드를 삽입한후 recoverFiles()를 호출합니다. 자신 하위 바로 아래 디렉토리 및 파일을 모두 탐색햇다면, 링크드 리스트 맨앞에 노드를 pop 함으로써, 자신을 불렀던 recoverFiles()로 돌아가는 dfs 형식으로 돌아가는 함수입니다.

2. recoverFile()

자신과 같은 <FILENAME> 을 가진 백업 파일들의 링크드 리스트를 인자로 받아서, 선택창을 통해 선택을 한후,

-n <NEWNAME> 을 통해 새로운 NEWNAME을 받았다면 그 경로로 아니라면 원래 <FILENAME> 의 경로로

선택한 백업 파일을 복원합니다.

// hash function

1. getMd5Sum() : 인자로 넘겨받은 filename 경로에 대한 md5 해시값을 만들어 md5Hash 배열로 넘겨줍니다.

2. getSha1Sum() : 인자로 넘겨받은 filename 경로에 대한 sha1 해시 값을 만들어 sha1Hash 배열로 넘겨줍니다.

3. 상세 설계

- ssu\_backup.c

메인이 되는 프로그램 c 코드입니다. 해당 c 파일에서는 사용자가 프로그램을 시작하기전 사용할 해시 함수를 인자로 받으며 시작을 하고, “20192392>”를 출력하며, 내부 명령어들의 실행을 받을 입력 대기 상태가 됩니다.

백업 프로그램 실행 시, 홈 디렉토리 하위에 /backup 디렉토리를 생성하고, 이미 있다면, 새로 생성 하지 않습니다.

입력 대기 상태에서 fget()를 통해 입력을 받고, strotok()을 통해 받은 입력을 “ “ 빈칸을 기준으로 토큰화 하여서, args[0] 이 정의된 명령어인지 확인합니다.

각 내부 명령어를 받은 후, (exit 명령어를 제외한) 모든 내부 명령어는 fork() execv()를 활용하여, 내부 명령어를 수행합니다. 이때, ssu\_backup 프로그램은 부모 프로세스이며, 자식 프로세스가 일하는 동안은 waitpid() 함수를 통해 자식 프로세스가 종료되기를 기다립니다. Exit 명령어의 경우에는 자식 프로세스를 부르기 전에 while() 문을 벗어난 후, exit(0)를 통해 프로그램을 정상적으로 종료 시키며, 그 외, add , remove, recover 함수에 경우에는 내부 적으로 해시함수를 사용하기 때문에 execv()의 2번째 인자인 args 배열에서 배열 마지막에 md5 나 sha1 문자열을 받아서 execv()를 진행합니다.

add, remove , recover 또한 각각 execv()를 통해 실행파일을 실행하며 인자로 fget()를 통해 받은 문자열을 token 화 한, args[]를 인자로 가져갑니다.

ls , vi , vim에 경우에는 각각 , “/bin/ls” , “/usr/bin/vi” , “/usr/bin/vim”을 첫번째 인자로 두번째 인자로 args를 받고 execv() 합니다. 그 외, 정의되지 않은 명령어의 경우와 “help”를 입력 받았을 경우에는 “help.c”에 실행파일인 “./help”를 부릅니다. 그 외, 정의되지 않은 명령어를 처리할 때 엔터 즉, 개행 문자를 단일로 받았다면 프롬프트를 재 출력하고, 다시 입력 대기 상태가 됩니다.(엔터 만 받았을 경우는 아무런 출력없음.)

차트이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

- add.c

도표이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

add.c 에서는 내부 명령어 add를 처리합니다. ssu\_backup\_u.c : getBackupDirectory()를 통해 사용자의 홈 디렉토리 하위 /backup 디렉토리의 경로를 받아옵니다. Getopt()를 통해 옵션을 처리하고 add에서는 -d 옵션 외의 – 옵션은 적절하지 않은 옵션으로 처리합니다. Argc ==2의 경우, ssu\_backup 프로그램에서 add 입력과 함께 md5 | sha1 문자열을 넘겨준 것으로, <FILENAME> 에 대한 입력이 없는 것을 의미합니다. 두 경우를 처리한후, ssu\_backup\_u.c : getProcessPath()

현재 입력된 경로에 ‘.’ 와 ‘..’ 와 ‘~’를 처리합니다. getProcessPath()에 절대 경로를 넣기 전, <FILENAME> 이 상대경로라면 현재 작업 디렉토리를 붙여 준 후 , getProcessPath()를 호출합니다. Lstat()를 통해 현재 경로의 파일이 일반 파일인지 디렉토리인지 검사한후, realpath()에 절대 경로를 넣어, 경로에 대한 권한이 있는지 경로가 제대로 된 경로인지 한 번 더 확인합니다. 그후, 입력된 경로가 4096 bytes를 넘는지 검사합니다. Absolute\_path 경로의 문자열 길이가 4096+1 로 선언되어 있기 때문에 문자열의 길이가 4096인지 확인하고, 4096이라면 마지막 문자가 ‘\n’ 이 아니라면 넘은 것으로 판단합니다.

입력경로가 ~를 벗어나는지 홈 디렉토리 하위 /backup 디렉토리 내 파일 혹은 디렉토리를 포함하는지 여부는

String.h : Strncmp() 를 통해 앞 문자열을 PATH\_HOME\_DIRECTORY 길이 만큼 비교하여 같은지 비교하고, string.h: strstr() 함수를 통해 /backup 디렉토리의 경로가 포함되는지를 확인합니다.

그후, linked\_list.c에 backup관련 함수 호출을 위해, 디렉토리 링크드 리스트를 만든 후, 디렉토리 백업 인 경우, backupFiles()를 호출하거나, 파일을 백업 하는 경우 backupFile()를 호출합니다.

- remove.c

도표이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

remove.c 에서는 내부 명령어 remove를 처리합니다. ssu\_backup\_u.c : getBackupDirectory()를 통해 사용자의 홈 디렉토리 하위 /backup 디렉토리의 경로를 받아옵니다. Getopt()를 통해 옵션을 처리하고 remove에서는 -a 옵션 과 -c 옵션 외의

– 옵션은 적절하지 않은 옵션으로 처리합니다. Argc <=2의 경우, ssu\_backup 프로그램에서 add 입력과 함께 md5 | sha1 문자열을 넘겨준 것으로, <FILENAME> 에 대한 입력이 없는 것을 의미합니다.

이때 -a -c 옵션 두개를 모두 받은 경우나, -c. 옵션을 받은 후, argc != 3 이면 에러 처리를 한다.

add, remove ,recover 에서는 execv()를 하기 전 마지막 인자로 md5 | sha1을 넣어주었기 때문에

remove -c <md5|sha1> 이 되어 인자가 3개인지 체크하는 것입니다.

c 옵션을 받았다면 위의 예외 처리를 거친 후 remove -c 입력을 받은 것이므로, removeAll() 을 호출합니다.

그 외의, 파일 및 디렉토리 삭제의 경우는,ssu\_backup\_u.c : getProcessPath() 현재 입력된 경로에 ‘.’ 와 ‘..’ 와 ‘~’를 처리합니다. getProcessPath()에 절대 경로를 넣기 전, <FILENAME> 이 상대경로라면 현재 작업 디렉토리를 붙여 준 후 , getProcessPath()를 호출합니다. Lstat()를 통해 현재 경로의 파일이 일반 파일인지 디렉토리인지 검사한후, realpath()에 절대 경로를 넣어, 경로에 대한 권한이 있는지 경로가 제대로 된 경로인지 한 번 더 확인합니다. 그후, 입력된 경로가 4096 bytes를 넘는지 검사합니다. Absolute\_path 경로의 문자열 길이가 4096+1 로 선언되어 있기 때문에 문자열의 길이가 4096인지 확인하고, 4096이라면 마지막 문자가 ‘\n’ 이 아니라면 넘은 것으로 판단합니다.

입력경로가 ~를 벗어나는지 홈 디렉토리 하위 /backup 디렉토리 내 파일 혹은 디렉토리를 포함하는지 여부는

String.h : Strncmp()를 통해 앞 문자열을 PATH\_HOME\_DIRECTORY 길이 만큼 비교하여 같은지 비교하고, string.h: strstr() 함수를 통해 /backup 디렉토리의 경로가 포함되는지를 확인합니다.

opendir을 통해 DIR \*dirp1, dirp2에 값을 받으려 하는데

dirp1의 경우는 백업 경로 그 자체를 열려고 시도하는 것이고, dirp2의 경우는 백업 경로 상위 디렉토리의 경로를 열려고 시도하는 것입니다.(백업된 파일이 존재한다면, 원본 파일이 존재하지 않더라도 remove를 할 수 있게 하기 위함.)

dirp1이 열렸다는 것은, 백업경로가 디렉토리 경로라는 것이므로, 디렉토리 하위 파일 및 디렉토리를 지우기 위해서,

-a 옵션을 받았는지 여부를 체크한후 linked\_list.c에 remove관련 함수 호출을 위해, 디렉토리 링크드 리스트를 만든 후, removeFiles()를 호출합니다.

dirp1이 NULL인 것을 확인했다면, dirp2를 백업 경로 상위 디렉토리를 열어 제거할 파일이 존재하는지 파일 링크드 리스트를 만들어 확인합니다. 이때, dirp1 과 dirp2 둘 모두 NULL이라면 Usage 출력합니다. 그후, dirp1 과 dirp2 가 열렸더라도, dirp2를 열어 얻은 DIR 포인터를 readdir()를 통해 열어 제거할 파일들을 넣는 파일 링크드 리스트가 비어 있다면,

제거할 파일이 존재하지 않는 것이므로, 이때 또한 Usage를 출력합니다.

제거할 파일이 존재하고 -a 옵션을 받았다면 모든 파일을 제거하고, -a 옵션을 받지 못했다면

제거할 파일이 여러 개 존재할 경우 선택 창을 통해 지울 파일을 정한 후, 해당 파일을 제거합니다. 이때 0번을 받았다면 아무 파일도 제거하지 않고 끝냅니다. 제거할 파일이 1개 일 경우는 선택 창을 출력하지 않고 바로 해당 파일을 제거합니다.

- recover.c

차트이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

recover.c 에서는 내부 명령어 recover를 처리합니다. ssu\_backup\_u.c : getBackupDirectory()를 통해 사용자의 홈 디렉토리 하위 /backup 디렉토리의 경로를 받아옵니다. Getopt()를 통해 옵션을 처리하고 remove에서는 -d 옵션 과 -n 옵션 외의

– 옵션은 적절하지 않은 옵션으로 처리합니다. Argc <=2의 경우, ssu\_backup 프로그램에서 add 입력과 함께 md5 | sha1 문자열을 넘겨준 것으로, <FILENAME> 에 대한 입력이 없는 것을 의미합니다.

-n 옵션을 받은 후, 뒤 인자가 해시 함수<md5|sha1> 외의 다른 인자가 없다면 , Usage를 출력합니다.

ex ) recover 1.txt -n

remove.c 와 recover.c에서 예외를 처리하는데 가장 큰 차이점은

-n 옵션을 받았을 경우, <FILENAME> 에 대해서 처리했던 모든 예외 처리를 <NEWNAME> 에 대해서도 처리해야 한다는 것입니다. 해당 순서도에 나타난 예외 처리가 remove.c 와 같지만, 내부에서는 flag\_n 이 1이라면 해당 예외처리를 2번씩 처리하고 있습니다.

그 외의, 파일 및 디렉토리 복원의 경우는,ssu\_backup\_u.c : getProcessPath() 현재 입력된 경로에 ‘.’ 와 ‘..’ 와 ‘~’를 처리합니다. getProcessPath()에 절대 경로를 넣기 전, <FILENAME> 이 상대경로라면 현재 작업 디렉토리를 붙여 준 후 , getProcessPath()를 호출합니다. Lstat()를 통해 현재 경로의 파일이 일반 파일인지 디렉토리인지 검사한후, realpath()에 절대 경로를 넣어, 경로에 대한 권한이 있는지 경로가 제대로 된 경로인지 한 번 더 확인합니다. 그후, 입력된 경로가 4096 bytes를 넘는지 검사합니다. Absolute\_path 경로의 문자열 길이가 4096+1 로 선언되어 있기 때문에 문자열의 길이가 4096인지 확인하고, 4096이라면 마지막 문자가 ‘\n’ 이 아니라면 넘은 것으로 판단합니다.

입력경로가 ~를 벗어나는지 홈 디렉토리 하위 /backup 디렉토리 내 파일 혹은 디렉토리를 포함하는지 여부는

String.h : Strncmp()를 통해 앞 문자열을 PATH\_HOME\_DIRECTORY 길이만큼 비교하여 같은 지 비교하고, string.h: strstr() 함수를 통해 /backup 디렉토리의 경로가 포함되는지를 확인합니다.

위의 내용을 -n <NEWNAME> 입력이 있다면, 동일하게 처리합니다.

opendir을 통해 DIR \*dirp1, dirp2에 값을 받으려 하는데

dirp1의 경우는 백업 경로 그 자체를 열려고 시도하는 것이고, dirp2의 경우는 백업 경로 상위 디렉토리의 경로를 열려고 시도하는 것입니다.(백업된 파일이 존재한다면, 원본 파일이 존재하지 않더라도 recover를 할 수 있게 하기 위함.)

dirp1이 열렸다는 것은, 백업경로가 디렉토리 경로라는 것이므로, 디렉토리 하위 파일 및 디렉토리를 복원하기 위해서,

-d 옵션을 받았는지 여부를 체크한후 linked\_list.c에 recover관련 함수 호출을 위해, 디렉토리 링크드 리스트를 만든 후, recoverFiles()를 호출합니다.

dirp1이 NULL인 것을 확인했다면, dirp2를 백업 경로 상위 디렉토리를 열어 복원할 파일이 존재하는지 파일 링크드 리스트를 만들어 확인합니다. 이때, dirp1 과 dirp2 둘 모두 NULL이라면 Usage 출력합니다. 그후, dirp1 과 dirp2 가 열렸더라도, dirp2를 열어 얻은 DIR 포인터를 readdir()를 통해 열어 복원할 파일들을 넣는 파일 링크드 리스트가 비어 있다면,

복원할 파일이 존재하지 않는 것이므로, 이때 또한 Usage를 출력합니다.

복원할 파일이 존재한다면 , recoverFile()를 호출합니다.(파일 링크드 리스트내에 노드(node\_f) 존재.)

recoverFiles()와 recoverFile() 모두 -n 옵션 처리를 위해 함수 프로토타입 상에서 char\* filename\_new 인자를 받습니다.

remove의 경우는 -c 나, -a 옵션을 받으면 하위 백업 파일은 모두 제거하므로, 처음에 백업파일이 여러 개인 파일을 제거 하려고 시도 할 때의 경우만 선택창을 출력하지만, 복원할 경우에는 현재 탐색중인 경로내에 같은 백업파일이 여러 개 있을 때마다 선택창을 출력해야 하므로, recoverFiles() 내부에서는 백업 파일 중에 <FILENAME> + timeString(timeString : \_YYMMDDHHMMSS)이 붙은 형태의 모든 백업파일을 링크드리스트로 만들어서 recoverFile()를 호출합니다.

- linked\_list.c

// add

- void backupFiles(Node \*\*head, int flag\_md5, int flag\_sha1); // backup files(BFS)

도표이(가) 표시된 사진

자동 생성된 설명

backupFiles()라는 함수에서는 인자로 받은 디렉토리 링크드 리스트가 NULL이 될때까지 탐색을 합니다.

현재 노드가 NULL이 아니라면, 해당 노드의 DIR 포인터를 readdir()를 통해 하위 파일 및 디렉토리 리스트를 모두 읽고, readdir()이 NULL이 되어 모두 다 읽었을경우, pop\_back()을 하여 해당 디렉토리 노드를 링크드 리스트에서 제거한후, 다음 노드가 있을 시, 다음 노드를 탐색합니다.

readdir() 하는 동안 하위 디렉토리를 발견하면 해당 디렉토리 경로를 통해 새로운 노드를 만들고 링크드 리스트 맨 뒤에 push\_back() 함으로써, BFS 탐색을 통해 처음 head의 디렉토리 노드에 대한 모든 하위 파일 및 디렉토리를 탐색할 수 있습니다.

- void backupFile(char \*filename, int flag\_md5, int flag\_sha1); // backup file

도표이(가) 표시된 사진

자동 생성된 설명

backupFile() 에서는 입력으로 받은 filename에 대해서 백업을 실시합니다.

flag\_md5 | flag\_sha1 중 플래그 값이 1인 해시 함수를 통해 백업된 파일이 존재한다면, “is already backuped” 을 출력 하도록 하며, 백업된 동일한 해쉬값의 파일이 존재하지않는다면, 백업 경로에 현재 파일명에 timeString을 붙힌 형태로 원본파일을 백업경로에 백업합니다. fork() execvp()를 통해 자식 프로세스가 cp 명령어를 수행하며(system() 사용하지 않음.), 그동안 부모 프로세스는 waitpid()를 통해 대기합니다.

// remove

- void removeAll(int flag\_c); // remove /backup directory

도표이(가) 표시된 사진

자동 생성된 설명

removeAll() 에 경우 remove.c 에서 remove -c 입력을 받았을 때 호출합니다. 별도의 경로나 링크드리스트를 받지 않은 이유는 ~/backup 디렉토리 하위 모든 파일과 서브 디렉토리를 지울 것이므로, 경로가 정해져 있기 때문입니다. flag\_c 가 1일 경우 동작하며, removeAll() 함수 내에서 /backup 디렉토리의 경로로 링크드리스트를 만든 후, removeFiles()를 호출합니다. 호출하기전, 몇 개의 파일과 몇 개의 서브 디렉토리를 지웠는지 출력하기 위하여 deleteF\_count 와 deleteD\_count int형 변수를 선언 후 인자로 넘겨주어, 하위 removeFiles()를 통해 몇 개의 파일과 디렉토리가 지워졌는지 카운트합니다.

- void removeFiles(Node \*\*head, int flag\_c, int \*deleteF\_count, int \*deleteD\_count); // directory remove(DFS)

도표이(가) 표시된 사진

자동 생성된 설명

removeFiles()의 경우는 DFS 형식입니다. 현재 head 노드가 가리키는 노드를 readdir()를 통해 읽고,

읽은 dirent 구조체의 타입이 디렉토리이면, 해당 디렉토리에 대한 노드를 만들어 head에 push\_front() 한 후, removeFiles()를 다시 부릅니다. 해당 removeFiles()는 헤드가 가리키는 노드에 대해서 탐색을 모두 한 후, 헤드의 노드를 pop\_front() 함으로써, 자신을 호출한 removeFiles()에 돌아갔을 때는 해당 디렉토리의 하위 모든 파일과 디렉토리를 삭제한후 돌아간것입니다.

- void removeFile(Node\_f \*\*head, int flag\_c, int count); // file remove

도표이(가) 표시된 사진

자동 생성된 설명

removeFile()에서는 파일 링크드 리스트를 인자로 받았고, 순차 탐색하여 count번에 해당하는 파일에 대해서, 삭제를 진행합니다.(사용자가 선택창을 통해 선택한 파일 삭제 구현.)

add 때와 동일하게 fork() execvp()를 통해 rm 명령어를 수행하며, system() 함수를 사용하지 않았습니다. 부모 프로세스는 자식 프로세스가 일하는 동안 대기합니다. 동시에 여러 프로세스가 일하지 않습니다.

// recover

- void recoverFiles(Node \*\*head, int flag\_md5, int flag\_sha1, int flag\_n, char\* filename\_new); // directory recover(DFS)

도표이(가) 표시된 사진

자동 생성된 설명

앞선, remove , add 와 달리 방문처리를 통해 백업 파일들을 관리합니다.

10개의 dirent 구조체 파일 리스트가 있다 가정하고, 같은 파일명의 여러 해시 값이 다른 파일들이 있다고 가정해보면, 일반적으로 구현하면, 뒤에서 해당 파일을 만났을 경우, 앞서서 해당 파일과 같은 파일명의 복원 리스트를 만들 때도 만들고, 현재 파일의 복원 리스트를 만들 때도 사용되므로, 백업된 같은 파일이 여러 개이면 동일한 선택창도 여러 번 출력 되는 문제가 있습니다. 이를 막기위하여 한번 같은 백업 파일들은 묶어서 파일 링크드 리스트를 만들어 방문 처리를 하며, 현재 탐색중인 파일과 같은 이름인지, 파일의 타입이 일반파일인지를 검사 후, 파일 링크드 리스트에 추가합니다.

현재 파일과 같은 백업 파일이 있는지 탐색을 마치면 rewinddir()를 통해 다시 탐색을 할수있도록 합니다. 방문처리를 했으므로, 앞서서 탐색했던 dirent 구조체 정보들은 무시할 수 있습니다.

remove때와 마찬가지로 DFS 형식이며, 디렉토리 타입을 만났을 경우, 해당 디렉토리를 새로운 노드로 만들어 head가 가리키게 하며 push\_front() 하여 recoverFiles()를 호출합니다.

- void recoverFile(Node\_f \*\*head\_f, int flag\_md5, int flag\_sha1, char\* filename\_new); // file recover

도표이(가) 표시된 사진

자동 생성된 설명

백업 파일이 여러 개 있을 경우, 파일 링크드 리스트에 여러 노드가 연결되어 인자로 넘어오며,

1개의 파일 노드를 가진 링크드 리스트가 들어올 시에는 , -n 인자를 받아 filename\_new 가 있을 시 해당 경로로 복원하며, 없을 시 백업 된 파일을 만들 때 쓴 원본의 경로로 복원합니다.

여러 파일 노드를 가진 링크드 리스트가 들어오면, -n 인자를 받아 filename\_new 가 있는지 확인하며, 선택창을 통해 사용자가 원하는 백업파일을 복원할 수 있도록 고르도록 하며, -n 인자를 받아 filename\_new 가 있을 시 해당 경로로 복원하며, 없을 시 백업 된 파일을 만들 때 쓴 원본의 경로로 복원합니다.

이때, -n 인자가 있을 경우 복원하려는 경로상에 상위 디렉토리들이 존재하지 않을 경우에는 디렉토리를 생성합니다.

- void compareAndCopyFile(char\* source\_path, char\* dest\_path, int flag\_md5, int flag\_sha1);

도표이(가) 표시된 사진

자동 생성된 설명

복원하려고 정한 백업 파일의 source 경로와 파일을 복원하게 되는 dest 경로를 인자로 받으며, flag\_md5, flag\_sha1를 인자로 가지고 있어, 사용자가 ssu\_backup 프로그램 실행 시 사용할 해시 함수를 사용하여 백업 파일과 복원하려는 위치에 원본파일이 존재한다면, 비교하여 같은 파일인지 검사합니다.

복원하려는 위치에 원본파일과 백업 파일의 해시 값이 다르거나, 원본 파일이 없다면, 복원을 진행합니다.

이때 add때와 마찬가지로 cp 명령어를 사용하며, fork() execvp()를 통해 자식 프로세스를 불러 명령어를 수행합니다. system() 함수를 사용하지 않았습니다. 부모 프로세스는 자식 프로세스가 명령어를 수행하는 동안 대기 상태로 있습니다. 부모 프로세스는 자식 프로세스가 명령어를 수행하는 동안 대기 하여 일을 하지 않습니다.

4. 실행결과

- 모든 ssu\_backup내 명령어 실행은 <“/home/hanseung/P1”, “~/P1”> 작업 디렉토리 에서의 실행입니다.

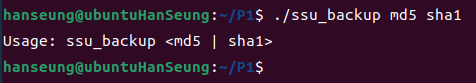
- 1. ssu\_backup

- <md5 | sha1> 에 해당하는 인자를 입력하지 않았거나, 적절한 인자를 입력하지 않았을 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 적절한 해시함수 인자를 주어도, 인자가 2개 이상인 경우 usage 출력 후 종료.



- md5 와 sha1 비교를 위한 해시 함수를 인자로 입력한 경우.(정상 실행 후 입력 대기)





- ssu\_backup 프로그램 처음 실행 시 홈 디렉토리 밑에 backup 디렉토리 성공적으로 생성.

텍스트이(가) 표시된 사진

자동 생성된 설명

- ssu\_backup 프로그램 처음 실행 시 홈 디렉토리가 밑에 이미 backup 디렉토리 있었을 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- ssu\_backup 프로그램 실행 후, 내장 명령어 입력 대기 상황 중 엔터(‘\n’)만 입력이 된 경우.(엔터 2번 입력.)

직사각형이(가) 표시된 사진

자동 생성된 설명

- ssu\_backup 프로그램 실행후, 내장 명령어 입력 대기 상황 정의되지 않은 명령어 입력이 된 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 2. 내장 명령어 : add

- 인자 입력이 없거나, 올바르지 않은 경로(해당 파일이나 디렉토리가 없을 경우 포함) 입력.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 입력 받은 경로가 ($HOME, ‘~’을 벗어나는 경우.) 및 일반 파일 이거나 디렉토리가 아니면 백업 안 하는 경우

- /dev/cdrom 과 /dev/sda는 각각, 링크 파일, 블록 파일

- 에러의 우선순위가 lstat()을 통한 (파일이 일반 파일, 디렉토리)가 우선순위이기 때문에 이와 같은 결과가 나왔습니다. “/etc/passwd”의 경우는 일반 파일이지만 ~을 벗어났기 때문에 can’t be backuped가 나온 모습.

- 에러 처리의 우선순위 : (파일이 일반 or 디렉토리) > ( ~를 벗어나거나 ‘~/backup’ 경로를 포함하는 경우.)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 입력 받은 경로가 ~/backup 디렉토리 혹은 백업 디렉토리 내 파일 및 디렉토리 포함하는 경우.

표준 출력 <입력받은 경로> can’t be backuped 로 출력된 결과. (~에 경우는 홈 디렉토리로 변환.)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

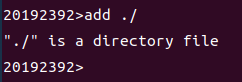
자동 생성된 설명

- 올바르지 않은 옵션이 들어왔을 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 첫번째 인자가 디렉토리 인 경우, -d 옵션이 들어오지 않았을 경우.



- 디렉토리일 경우 -d 옵션을 준 경우

텍스트이(가) 표시된 사진

자동 생성된 설명

- 이미 백업 한 파일이 존재한다면 백업을 진행하지 않는 경우.

- 파일

텍스트이(가) 표시된 사진

자동 생성된 설명

- 디렉토리

(위의 add.c , remove.c, recover.c를 이미 백업 하여 3개 파일은 already backuped 가 나온 모습.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- 이미 백업을 한 같은 이름의 파일이 존재하나, 해당 파일과 내용이 다른 경우 다시 백업.

- 즉 , 이미 add.c\_230327215259가 원본과 동일한 백업 파일이 존재하고, 수정을 하였습니다.

- 수정을 거친 후, 달라진 원본은 add.c\_230327215259 와 다르므로, 새로운 백업파일 add.c\_230327215412로

새로이 백업이 됩니다. 수정을 하지 않은 상태로 다시 백업을 시도하면, 이미 원본과 동일하기 때문에 백업이

안됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 디렉토리 내부에 디렉토리 와 파일이 존재하는 경우. (하늘색 파일 링크 파일이므로, 백업 하지 않습니다.)

- 이때, 백업할 파일이 존재하지 않는 디렉토리의 경우는 백업 디렉토리 하위에 디렉토리를 생성하지 않습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 3. 내장 명령어 : remove

- remove 명령어의 첫번째 인자가 없거나 올바르지 않은 경로 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

- 입력 받은 경로가 ($HOME 또는 ~)을 벗어난 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 입력 받은 경로가 백업 디렉토리 혹은 백업 디렉토리 내 파일 및 디렉토리를 포함하는 경우

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- 첫 번째 인자가 디렉토리인 경우 -a 옵션을 사용하지 않았다면 (에러 처리 후 프롬프트 재 출력.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- 디렉토리인 경우 -a 옵션을 정상적으로 부여하였을 경우. (정상 적인 삭제가 이루어져야 함.)

내부적으로, 지정한 remove 명령어로 지정한 경로에 모든 하위 파일 및 서브 디렉토리를 삭제합니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove 명령어에 올바르지 않은 옵션이 들어온 경우. (Usage 출력 후, 프롬프트 재 출력.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove -a -c 옵션을 가지 사용하였을 경우. (Usage 출력 후, 프롬프트 재 출력.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove -c 옵션 중 다른 인자를 입력하였을 경우.(Usage 출력 후, 프롬프트 재 출력.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove -c 입력이 정상적으로 이루어졌을 경우.(1. 하위 파일이나 서브 디렉토리가 존재할 경우.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- remove -c 입력이 정상적으로 이루어졌을 경우(2. 하위 파일이나 서브 디렉토리가 존재하지 않을 경우. )

텍스트이(가) 표시된 사진

자동 생성된 설명

- 백업된 파일 삭제(여러 백업 파일 존재)

텍스트이(가) 표시된 사진

자동 생성된 설명

- 백업된 파일 삭제 -a 옵션.(여러 백업 파일 존재)

텍스트이(가) 표시된 사진

자동 생성된 설명

- add 내부 명령어를 통해 normal2 디렉토리를 백업 후,

Remove 내부 명령어를 통해 모두 삭제.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 4. 내장 명령어 : recover

- recover 첫번째 인자 입력이 없는 경우 Usage 출력 후 프롬프트 재 출력.

텍스트이(가) 표시된 사진

자동 생성된 설명

- recover 올바르지 않은 경우(백업 파일이나 백업 된 디렉토리 없을 경우 도 포함.) Usage 출력 후 프롬프트

- add 명령어를 통해 백업한후, remove 명령어를 통해 백업 파일을 지워 read에 대한 백업 파일이 없는 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 첫번째 인자 <FILENAME> 혹은 -n 옵션 뒤에 <NEWNAME> ($HOME 또는 ~)를 벗어 난 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 첫번째 인자 <FILENAME> 혹은 -n 옵션 뒤에 <NEWNAME> /backup 디렉토리를 벗어 난 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 디렉토리를 recover 하는 경우 ‘-d’ 옵션을 주지 않은 경우.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 올바르지 않은 옵션이 들어온 경우. Usage 출력 후 프롬프트 재 출력.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 백업파일의 기존 경로가 <FILENAME>과 같고 같은 해시 값을 가지는 경우.

add 하자마자 바로 recover를 시도하면 백업파일과 원본이 동일한 것을 이용하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 백업 파일이 있지만 , 원본 파일이 없는 경우에도 정확한 경로를 줌으로서, 정상적으로 복원.

- ~/test/1.c 라는 파일을 백업 후, 원본을 삭제한 후, 복원을 시도. 정상적으로 복원된 실행 결과.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 복원하려고 하는데 백업된 파일이 여러 개 인 경우.(해시 값이 다른 같은 이름의 파일.)

텍스트이(가) 표시된 사진

자동 생성된 설명

- 재귀적으로 디렉토리를 복원하는 도중, 같은 백업 파일이 여러 개 있을 경우.

1.c의 백업이 2개 존재하므로, 복원을 재귀적으로 할 때 3번 시도하며 그중 1번은 선택창을 출력합니다.

고른 1번의 1.c 백업이 기존 원본과 달랐기 때문에 정상적으로 복원이 되었고, 나머지 2개의 파일은

동일한 해시 값을 가진 파일이었으므로, same file임을 출력하고 복원하지 않았습니다.

도표이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

- n 옵션의 실행 결과의 테스트 케이스를 위한 트리 구조.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

- -n <NEWNAME> 파일 복원

텍스트이(가) 표시된 사진

자동 생성된 설명

- -d -n <NEWNAME> 옵션을 주어 디렉토리 복원.

텍스트이(가) 표시된 사진

자동 생성된 설명

- 5. 내장 명령어 : ls, vi(m)

- ls 명령어와 ls 명령어의 옵션 처리까지 가능한 모습

텍스트이(가) 표시된 사진

자동 생성된 설명

- vi 1.c 로 새로운 파일 열기.





- vim 11.c 로 새로운 파일 열기.

직사각형이(가) 표시된 사진

자동 생성된 설명

- 6. 내장 명령어 : help

- 프롬프트 상에서 help 입력

텍스트이(가) 표시된 사진

자동 생성된 설명

- 7. 내장 명령어 : exit

- 프롬프트 상에서 exit 입력

텍스트이(가) 표시된 사진

자동 생성된 설명