벤처중소기업학과 20171969 박건호

1. 과제개요 (구현한 내용 요약)

ssu\_backup.c

가장 먼저 실행하는 프로세스.

인자로 md5 혹은 sha1을 받아 환경변수로 저장, 자식프로세스에서 참고하여 해시 사용하였음.

실행시 홈디렉토리에 "backup"파일이 없으면 생성함.

20171969> 출력 후 사용자 입력 대기

사용자 입력을 띄어쓰기로 구분하여 문자 파싱하고 execvp()함수에 인자로 건네줌

첫 인자에 따라 내장 명령어 수행

지정된 내장 명령어 이외 입력시 help 출력, exit 입력시에 execvp하지않고 반복문 빠져나가 프로그램 종료함.

add.c

인자로 입력받은 파일을 백업 디렉토리에 복사함.

입력받은 파일이 디렉토리일 시에는 -d옵션 사용해야 하고, 내부를 재귀적으로 탐색하여 파일(디렉토리 제외)이 있다면 복사함.

파일 복사시에 파일이 위치한 절대경로까지의 디렉토리가 존재하지 않는다면 디렉토리를 만드는 보조함수를 구현하였음.

remove.c

인자로 입력받은 파일을 백업디렉토리로부터 삭제함.

백업된 파일들은 백업시간이 파일 이름에 저장되어 있는데, 백업시간을 제외한 파일이름이 같은 파일을 백업된 파일이라고 판단함. (물론 존재하는 경로가 동일해야함)

존재 경로가 동일하다는 것은

/home/oslab/P1/a/a.txt 파일은 홈 디렉토리를 제외하면 경로가 /P1/a/a.txt 이고...

/home/oslab/backup/P1/a/a.txt\_백업시간 파일은 홈 디렉토리/backup을 제외하면 경로가 /P1/a/a.txt\_백업시간

이기에... 존재 경로가 /P1/a로 같다는 것

옵션 사용이 없다면 존재 경로가 같은 파일의 목록을 보여주고 사용자의 입력으로 삭제할 파일을 선택하도록 구현

-a 옵션을 사용한다면 목록을 보여줄 필요 없이 전부 다 삭제하도록 구현

-c 옵션을 사용한다면 무조건 >remove -c 로 입력해야하고... backup 디렉토리 내부의 모든 파일과 디렉토리를 재귀적으로 삭제 후 각각 몇개가 삭제되었는지 출력

recover.c

인자로 입력받은 파일을 백업디렉토리로부터 백업.

인자는 현재 작업 경로 기준이지만 받았을때 백업디렉토리 기준으로 바꿔준다.

/home/oslab/P1/a/a.txt > /home/oslab/backup/P1/a/a.txt

파일을 백업할때 경로가 같은 파일이 여러개 존재한다면 같은 파일의 목록을 보여주고 사용자의 입력으로 백업할 파일을 선택하도록 구현

옵션 -d 는 디렉토리일 경우에만 유효하게 구현

입력한 디렉토리 기준으로 백업디렉토리 기준에서 하위 모든 파일을 재귀적으로 백업.

백업대상은 오직 정규파일이며, 디렉토리를 백업하는 경우는 그 디렉토리 아래에 정규파일이 있는 경우이다.

옵션 -n 사용시 다음 인자로 무조건 저장하고자 하는 새로운 파일이름이 주어져야하고 그 경로로 백업위치가 결정된다. - 디렉토리, 정규파일 둘 다 동일하게 적용

따라서 -d 옵션, -n옵션 동시 사용도 가능하다

help.c

printf()로 Usage 출력하는 프로그램

2. 기능 (구현한 기능 요약)

ssu\_backup.c

void print\_help(); // Usage 출력

모든 내장명령어를 실행할 수 있는 환경을 구축.

내장명령어는 add remove recover ls vi vim help exit 가 있다

명령어를 사용자로부터 입력받기 전에 20171969>를 출력하여 입력 대기중이라는 것을 알려주며, 유효하지 않은 명령어나 옵션을 입력하면 Usage 를 출력하고 20171969> 를 다시 출력하여 사용자의 입력을 대기한다. 터미널 강제 종료가 아닌 이상 사용자는 exit를 입력함으로써 프로세스를 종료할 수 있다. 각 내장명령어의 실행이 끝나면 다시 2017969>를 출력하여 사용자의 입력을 대기한다.

add.c

int backup\_file(char \*src\_path); // 지정한 경로의 파일을 백업

void search\_directory(char \*src\_dir); // 지정한 경로의 디렉토리를 탐색

int file\_exists(const char \*path); // 파일이 존재하는지 체크

int is\_regular\_file(const char \*path); // 파일이 정규파일인지 체크

int is\_directory(const char \*path); // 파일이 디렉토리인지 체크

void create\_directory\_recursive(const char \*dir\_path); // 디렉토리를 생성하는데 필요한 최상위 디렉토리로부터 따라 올라오는 재귀성격의 함수

void create\_directory\_path(const char \*dir\_path); // 해당경로까지의 디렉토리를 없다면 생성

int get\_md5(char \*path, unsigned char \*result); // MD5 해시값을 반환하는 함수

int get\_sha1(char \*path, unsigned char \*result); // SHA1 해시값을 반환하는 함수

사용자로부터 파일의 상대경로나 절대경로를 입력받아 홈디렉토리 밑 백업디렉토리에 백업한다. 입력받는 파일의 경로는 파일일 경우에는 별도의 옵션이 필요 없지만, 디렉토리일 경우에는 -d 옵션을 반드시 추가해야 한다. 디렉토리를 -d 옵션 없이 add명령하면 실행이 되지 않는다. 파일의 백업이 끝나면 백업디렉토리 하위에 홈디렉토리를 기준으로 같은 경로상의 파일이 생긴다. 그 파일의 이름은 끝에 \_백업시간이 추가되어있어 언제 백업을 하였는지 파악할 수 있다. 디렉토리를 백업하는 경우 입력받은 경로의 디렉토리 안에 있는 모든 파일에 대해 재귀적으로 탐색, 백업한다.

remove.c

void remove\_files(char \*path, int all\_flag); // 지정한 경로의 파일들을 탐색하고 옵션에 따라 사용자의 입력대로 삭제 혹은 전부 삭제

void clear\_backup\_dir(char \*path, int \*file\_count, int \*dir\_count); // 지정한 디렉토리 내부를 재귀적으로 탐색하며 모든 파일과 디렉토리를 삭제

int file\_exists(const char \*path); // 파일이 존재하는지 체크

int is\_regular\_file(const char \*path); // 파일이 정규파일인지 체크

int is\_directory(const char \*path); // 파일이 디렉토리인지 체크

사용자로부터 파일의 상대경로나 절대경로를 입력받아 백업디렉토리에 존재하는 백업파일을 삭제한다. 백업디렉토리와 홈디렉토리를 기준으로 같은 경로에 있는 파일이 삭제의 대상이 되며, 옵션 없이 사용한다면 해당 파일이 정규파일일 시 어떤 파일을 제거할 것인지 사용자 입력으로 선택해야한다. 백업한 시간이 다른 같은 경로의 같은 이름을 가진 파일들이 존재할 수 있기 때문에 여러개의 파일중에서 어떤 파일을 삭제할 것인지 사용자가 결정해야 한다. 디렉토리를 삭제하기 위해선 -a 옵션을 사용해야한다. -a 옵션을 사용하면 여러개의 파일 중 하나를 고를 필요없이 탐색이 돈 모든 파일과 디렉토리를 삭제한다. 디렉토리를 입력한다면, 해당 디렉토리 안에 있는 파일과 디렉토리를 제거하는 과정을 수행한다. -c 옵션을 사용한다면, backup디렉토리 안에 있는 모든 파일과 디렉토리를 제거하고 그 개수를 출력한다.

recover.c

void recover\_file(char \*filename, char \*newfile, int recursive); // 리커버리할 백업디렉토리 기준의 파일경로를 받아서 저장하고자 하는 홈디렉토리 기준의 경로에 파일을 리커버리 하는 함수

void recover\_directory(char \*dirname, char \*newdir); // 디렉토리를 입력받아 대상 디렉토리를 탐색하며 원하는 사용자가 입력한 홈디렉토리 기준 경로에 재귀적으로 리커버리 하는 함수

void create\_directory\_recursive(const char \*dir\_path); // 디렉토리를 생성하는데 필요한 최상위 디렉토리로부터 따라 올라오는 재귀성격의 함수

void create\_directory\_path(const char \*dir\_path); // 해당경로까지의 디렉토리를 없다면 생성

int get\_md5(char \*path, unsigned char \*result); // MD5 해시값을 반환하는 함수

int get\_sha1(char \*path, unsigned char \*result); // SHA1 해시값을 반환하는 함수

backup디렉토리에 저장되어 있는 백업디렉토리로부터 사용자가 원하는 파일경로로 파일을 백업하는 작업을 수행한다. 사용자가 입력한 홈디렉토리 기준의 경로를 백업디렉토리를 기준으로 바꾸어 보았을 때, 그 안에 같은 경로상의 파일이 존재한다면, 사용자가 입력한 기준경로의 파일이 없더라도 리커버리를 수행한다. 만약 파일이 여러개일 경우에는 사용자로부터 입력을 받아 리커버리할 파일을 선택한다. 디렉토리를 리커버리 하기 위해선 -d 옵션을 반드시 부여해야 한다. -d 옵션 부여를 한다면 백업경로상 기준경로가 되는 디렉토리 하위에 존재하는 모든 파일을 재귀적으로 전부 리커버리 한다. 이때, 여러개의 동일경로 파일이 존재한다면 사용자로부터 입력을 받아 각각 어떤 파일로 리커버리 할 것인지 결정받는다. 만약 -n 옵션을 사용한다면 사용자가 리커버리를 하고자 한 파일들이 -n 옵션 뒤에 입력으로 지정한 path에 저장이 된다. 만약 존재하지 않은 path였다면 해당 path까지 디렉토리를 재귀적으로 만들고 리커버리를 수행한다.

3. 상세설계 (함수 및 모듈 구성, 순서도, 구현한 함수 프로토타입)

ssu\_backup.c

인자로 argv[1] 에 md5나 sha1이 들어오지 않으면 에러 후 종료

들어온 인자값을 환경변수에 설정해서 자식프로세스에서 사용하도록 구현했음

현재 작업경로를 사용자 홈으로 이동해서 "backup"이라는 이름을 가진 백업 디렉토리가 있는지 확인하고 없다면 생성한다

환경변수 PATH를 가지고와서 : 현재작업디렉토리 (cwd)변수 를 추가함으로써... 현재 작업디렉토리에 있는 실행파일들을 모두 실행 가능하도록 구현했다

반복문을 통해 명령어 입력받기를 반복, 대기한다.

20171969>를 출력하고 사용자로부터 입력을 대기한다.

사용자로부터 들어온 입력을 공백문자 기분으로 나누어서 토큰화한다.

이 토큰들은 이어서 실행할 내장명령어의 이름과 옵션, <FILENAME> 같은 인자가 된다.

첫 인자를 비교하여 add 라면 fork()후 execvp()로 add 프로그램 실행

add 라면 fork()후 execvp()로 add 프로그램 실행

remove 라면 fork()후 execvp()로 remove 프로그램 실행

recover 라면 fork()후 execvp()로 recover 프로그램 실행

ls 라면 fork()후 execvp()로 ls 프로그램 실행

vi 혹은 vim 이라면 fork()후 execvp()로 vi 프로그램 실행

help 라면 fork()후 execvp()로 help 프로그램 실행

이외의 인자가 들어온다면 정의한 print\_help()를 실행하여 Usage를 출력한다.

exit를 입력한다면 반복문을 빠져나가 ssu\_backup 프로그램이 정상적으로 종료된다.

add.c

주요 기능 함수는 두가지이다.

int backup\_file(char \*src\_path); - 인자로 file의 절대경로를 받아서 백업한다

void search\_directory(char \*src\_dir); - 인자로 directory의 절대경로를 받아서 내부를 재귀적으로 탐색한다.

backup\_file 은 간단하게 정규파일 하나만을 백업하는 작업을 수행한다. 인자로 디렉토리가 들어오는 경우는 상정하지 않았으므로 주의하여 구현하였다.

search\_directory 은 디렉토리를 인자로 받아야만 한다. 디렉토리를 인자로 받으면 해당 디렉토리를 열어서 그 안에있는 파일들을 하나하나 다 확인한다. 그 파일이 정규파일이라면 backup\_file()를 통해 백업을 한다. 그 파일이 디렉토리라면 search\_directory를 통하여 다시 재귀적으로 내부를 탐색한다. 결론적으로 최초의 인자 디렉토리 하에 있는 모든 파일이 백업된다. 여기서 주의할 것은 디렉토리는 재귀적으로 탐색을 할 뿐 그 자체를 백업하는 것이 아니다. 확실한 백업 대상은 정규파일 이므로 만약 디렉토리 내부에 정규파일은 존재하지 않고 디렉토리만 무수히 많다면 디렉토리들은 백업이 되지 않는다.

backup\_file 내부에서 정규파일 백업 전에 호출하는

void create\_directory\_path(const char \*dir\_path); 함수가 있는데,

이 함수는 백업을 하는 파일이 위치할 디렉토리가 존재하지 않는다면 디렉토리를 재귀적으로 최상위 디렉토리까지 탐색하여 현 위치까지 존재하지 않는 디렉토리에 대해서만 디렉토리를 생성해주는 기능을 한다. 그렇기 때문에 파일을 백업할 시에만 디렉토리를 생성하게 되고, search\_directory는 따로 디렉토리를 백업하는 명확한 기능은 가지고 있지 않은 것이다. 이러한 종속 관계를 유념하여 구현하였다.

remove.c

주요 기능 함수는 두가지이다.

void remove\_files(char \*path, int all\_flag);

void clear\_backup\_dir(char \*path, int \*file\_count, int \*dir\_count);

remove\_files 함수는 제거하려는 파일의 절대경로를 인자로 받아 파일을 제거하는 기능을 수행한다. 기능 수행을 위해 입력받은 절대경로를 홈 디렉토리에 있는 backup이 포함되도록 변환하는 내부 과정을 거친다. 제거하려는 파일이 존재하는 디렉토리를 dirname() 라이브러리 함수를 통해 구하고, 그 위치의 디렉토리를 오픈하여 그 안의 파일들을 하나하나 확인한다. 그 이유는 사용자가 지정했던 삭제를 원하는 경로에 존재하는 파일이 여러개일 수 있기 때문이다. 백업파일은 경로가 같아도 수정여부에 따라 백업이 여러번 이루어질 수 있다. 파일을 탐색하며 정규파일만 확인을 하는데, 사용자가 지정했던 절대경로와 백업파일에서 홈/backup 디렉토리를 제외한 경로가 같은 모든 정규파일을 탐색 순서에 맞추어 출력해준다. 출력시에 천의 자리 숫자에 쉼표구분을 위해 locale.h 헤더를 포함하고 setlocale함수를 호출했다. 그리고 파일의 크기(stat.st\_size)를 출력할때 형식 지정자는 MAC OS BSD기반에서 현재 %lld를 사용하기 때문에 lld를 사용하였다. 현재 출력하는 과정은 -a 옵션이 없는 상태를 기반으로 설명한 것인데, 이 -a옵션이 켜지면 모든 파일은 경로가 같음에 상관 없이 전부 삭제를 하면 된다. 그래서 a옵션이 켜져있는 경우는 출력, a 옵션이 꺼져있는 경우는 단순 삭제 후 결과 출력을 하도록 구현하였다. 만약 삭제를 진행했다면 삭제 플래그를 온으로 바꾸어 -a옵션이 켜진 상태는 삭제 후에 바로 함수를 리턴하여 프로세스의 진행을 원할하게 한다. 하지만 출력만하고 아직 삭제하지 않은 -a옵션이 꺼진 상태에서는 다시 파일을 읽어야 하는데, 그 전에 사용자에게서 숫자를 입력받아 그 숫자번째의 파일을 삭제하도록 구현하였다. 파일을 삭제하는 이 함수와 다르게

void clear\_backup\_dir(char \*path, int \*file\_count, int \*dir\_count); 함수는 디렉토리를 기준으로 삭제를 진행한다. 다만 디렉토리 진행시에는 -a 옵션이 반드시 켜져있어야 한다. 디렉토리 기준 파일 삭제 진행시에는 디렉토리의 경로를 인자로 받아 그 디렉토리 하위에 있는 모든 파일들에 대해 삭제를 진행한다. 인자로 받은 디렉토리를 오픈해서 그 아래엤는 파일들을 하나씩 읽어가는데, 그것이 디렉토리라면 재귀적으로 void clear\_backup\_dir(char \*path, int \*file\_count, int \*dir\_count); 를 호출하여 그 밑의 파일들을 탐색, 삭제하고, 그것이 파일이라면 단순히 제거를 한다. 여기서는 add 프로그램과 다르게 탐색하며 발견한 모든 디렉토리까지 삭제를 진행한다. 하나의 디렉토리와 파일을 삭제할 때마다 각각의 카운트를 포인터 변수를 통해 1씩 증가시켜 결과적으로 main함수에서 몇개의 파일과 디렉토리를 제거했는지 파악할 수 있도록 한다. -c 옵션은 따로 경로를 입력받지 않고 remove -c오 입력을 받아 backup디렉토리 하위에 있는 모든 백업돈 파일을 삭제한다. 위의 알고리즘에 따르면 void clear\_backup\_dir(char \*path, int \*file\_count, int \*dir\_count); 함수에 첫 인자로 backup디렉토리를 건네주면 모든 하위 디렉토리, 파일을 제거하고 제거한 개수를 알아낼 수 있다.

recover.c

주요 기능 함수는

void recover\_file(char \*filename, char \*newfile, int recursive);

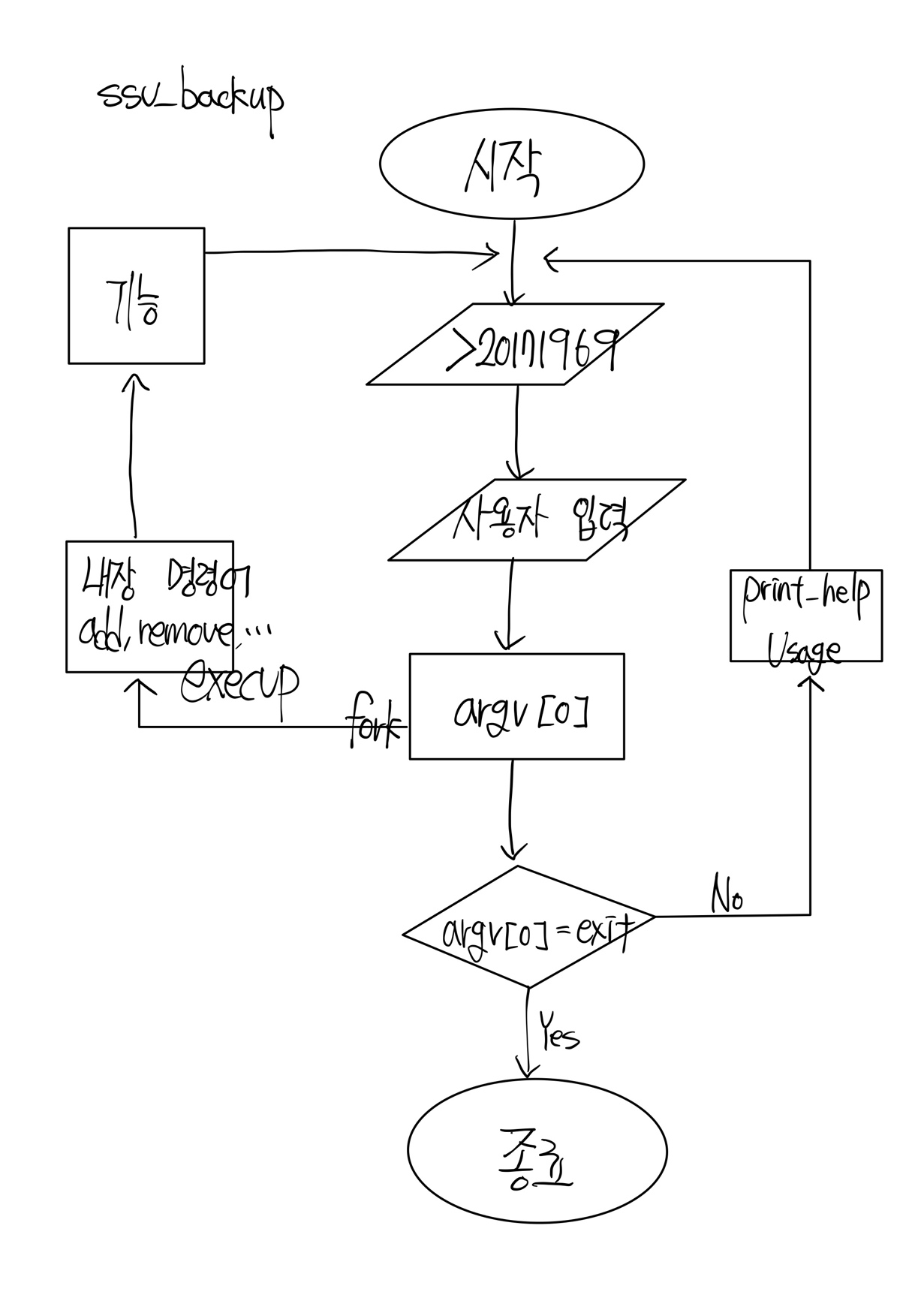
void recover\_directory(char \*dirname, char \*newdir);

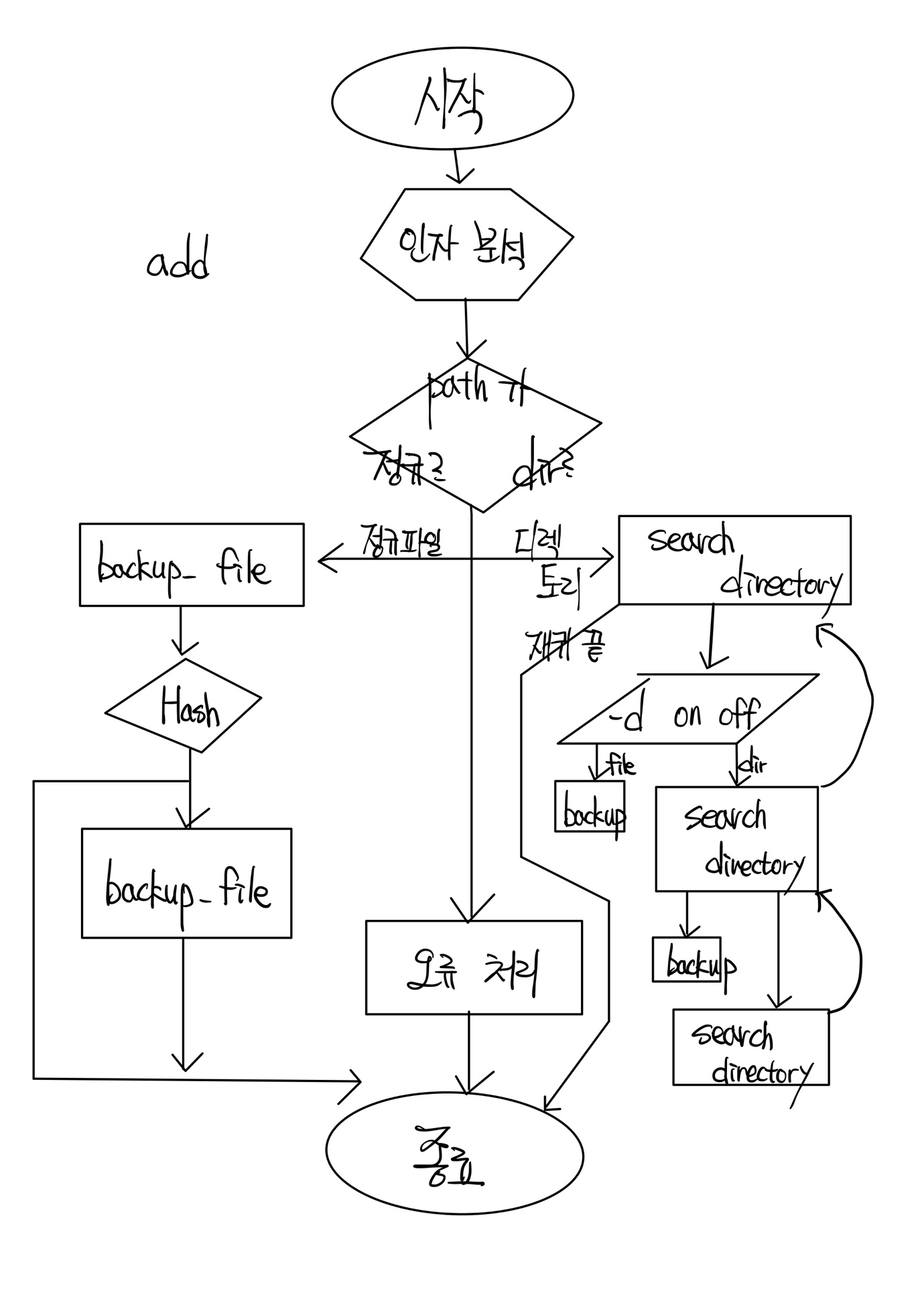
이 있다.

void recover\_file(char \*filename, char \*newfile, int recursive);

는 main이 인자로 받은 상대경로나 절대경로의 path를 backup디렉토리가 포함된 경로로 바꾸어서 첫번째 인자로 전달을 받는다. 두번째 인자는 리커버리 하고자 하는 백업된 파일이 어떤 절대경로에 (어떤 이름으로) 저장이 되는지에 대한 정보이다. 세번째 리컬시브는 옵션 -d 가 켜져있는지 안켜져있는지 확인하기 위해 구현을 했지만, main함수 부분에서 옵션을 처리하여 크게 중요하지는 않다. 디렉토리는 반드시 -d 옵션을 사용해야 한다. -d옵션을 사용하면 하위에 있는 모든 파일에 대해 리커버리를 진행하는 것이다. 다시 본론으로 돌아와, void recover\_file(char \*filename, char \*newfile, int recursive); 에 대해 자세히 보면, 백업하려는 파일이 들어있는 백업디렉토리 하부의 디렉토리를 오픈해서 그 안의 파일들을 살펴본다. 그 이유는 앞서 본 것 같이 백업된 파일은 수정여부에 따라 여러개가 있을 수 있기 때문에 사용자로부터 입력을 받아 어떤 파일로 리커버리 할 것인지 판단해야 하기 때문이다. 디렉토리에서 파일을 하나씩 읽어가다가 backup디렉토리를 제외한 모든 경로가 같은 - 즉 리커버리 대상인 파일이 있다면 파일의 정보와 백업시간을 출력한다. 출력시에는 파일카운트를 하나씩 증가시키는데, 모든 파일을 읽고 난 후에도 파일 카운트가 0이라면, 백업 대상 파일이 없다는 것이므로 메세지를 출력 후 리턴하여 프로세스 흐름을 원할하게 한다. 만약 파일이 1개 이상 있다면 사용자로부터 입력을 받아 번호에 해당하는 파일을 리커버리 한다. 다만 리커버리 전에 본래 경로에 있어야 하는 파일과 리커버리 하려는 파일을 해시함수를 통해 비교하여 동일한 파일이라면 리커버리 하지 않고 메세지 출력 후 함수를 리턴한다. 리커버리를 할 때에는 -n조건에 따른 부가적인 조치가 필요한데, -n NEWFILE로 입력받은 NEWFILE 경로는 파일이 실제로 존재하는 경로가 아니라. 결국 리커버리된 파일이 그 이름(절대경로)을 가진다고 생각하였다. 실제로 존재하는지 안하는지는, 백업디렉토리내부의 백업된파일 기준으로 찾아가서 찾으면 되는데, 사실 리커버리의 의미가 사라진 파일을 다시 불러올 수 있다는 것이기 때문에 본래경로의 파일을 존재하지 않아도 실행에 지장이 없다. 파일의 리커버리는 이 함수로 구현이 되었고 디렉토리를 리커버리하려는 경우에 대해 생각해보아야 한다.

void recover\_directory(char \*dirname, char \*newdir); 을 통해 재귀적인 방법으로 구현을 했는데, 우선 리커버리를 하고자 하는 백업디렉토리에서 인자로 받은 해당 디렉토리를 연다. 여기서 -d 옵션은 반드시 켜져 있어야 하며, -d 옵션이 켜져있기 때문에 내부에 있는 모든 파일과 디렉토리를 재귀적으로 리커버리한다. 디렉토리를 연 상대에서 안에 있는 파일을 하나하나 살펴보면서 만약 그 파일이 정규파일이라면 void recover\_file(char \*filename, char \*newfile, int recursive); 함수로 파일 리커버리를 시도한다. 이때, 파일이 여러개라면 어떤 파일로 리커버리할지는 그때그때 사용자에게서 입력받아서 결정을 한다. 만약 파일이 디렉토리라면 재귀적으로 void recover\_directory(char \*dirname, char \*newdir); 를 호출하여 그 아래에 있는 파일들에 대해서도 전부 리커버리를 시도한다.





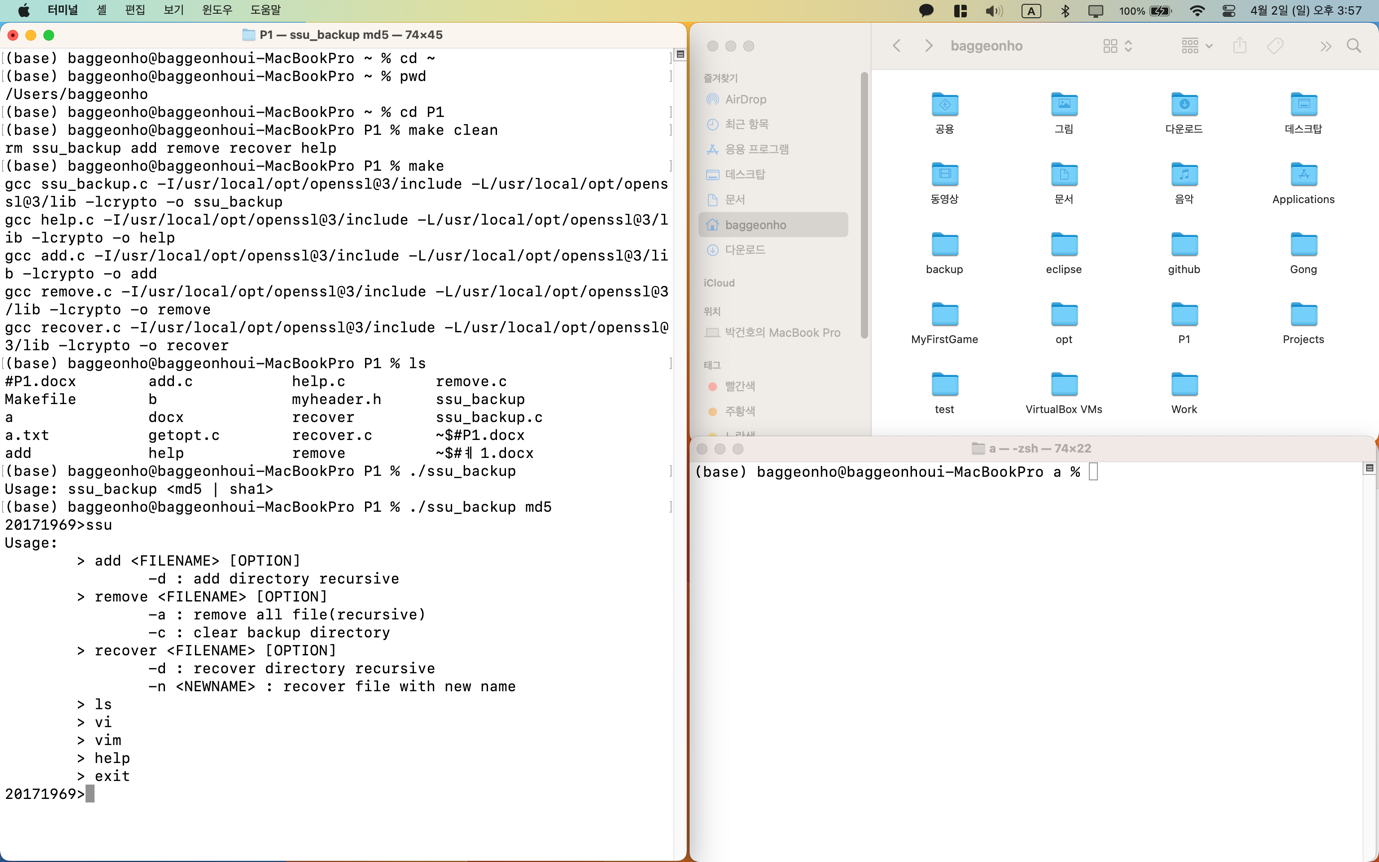
도표이(가) 표시된 사진

자동 생성된 설명

도표이(가) 표시된 사진

자동 생성된 설명

4. 실행결과



cd ~를 통해 홈 디렉토리로 이동

홈 디렉토리는 /Users/baggeonho

P1폴더에서 make 후 ssu\_backup 실행

정의되지 않은 명령어 입력 시 Usage: 출력

텍스트이(가) 표시된 사진

자동 생성된 설명

엔터 입력 시 프롬프트 재출력

add 형식에 맞지 않는 입력으로 인한 Usage 출력

현재 디렉토리 내부 파일 목록은 명세 그림 1과 같음.

/home/oslab/

┗ P1/

┣ a/

┃ ┣a.txt

┃ ┣ b/

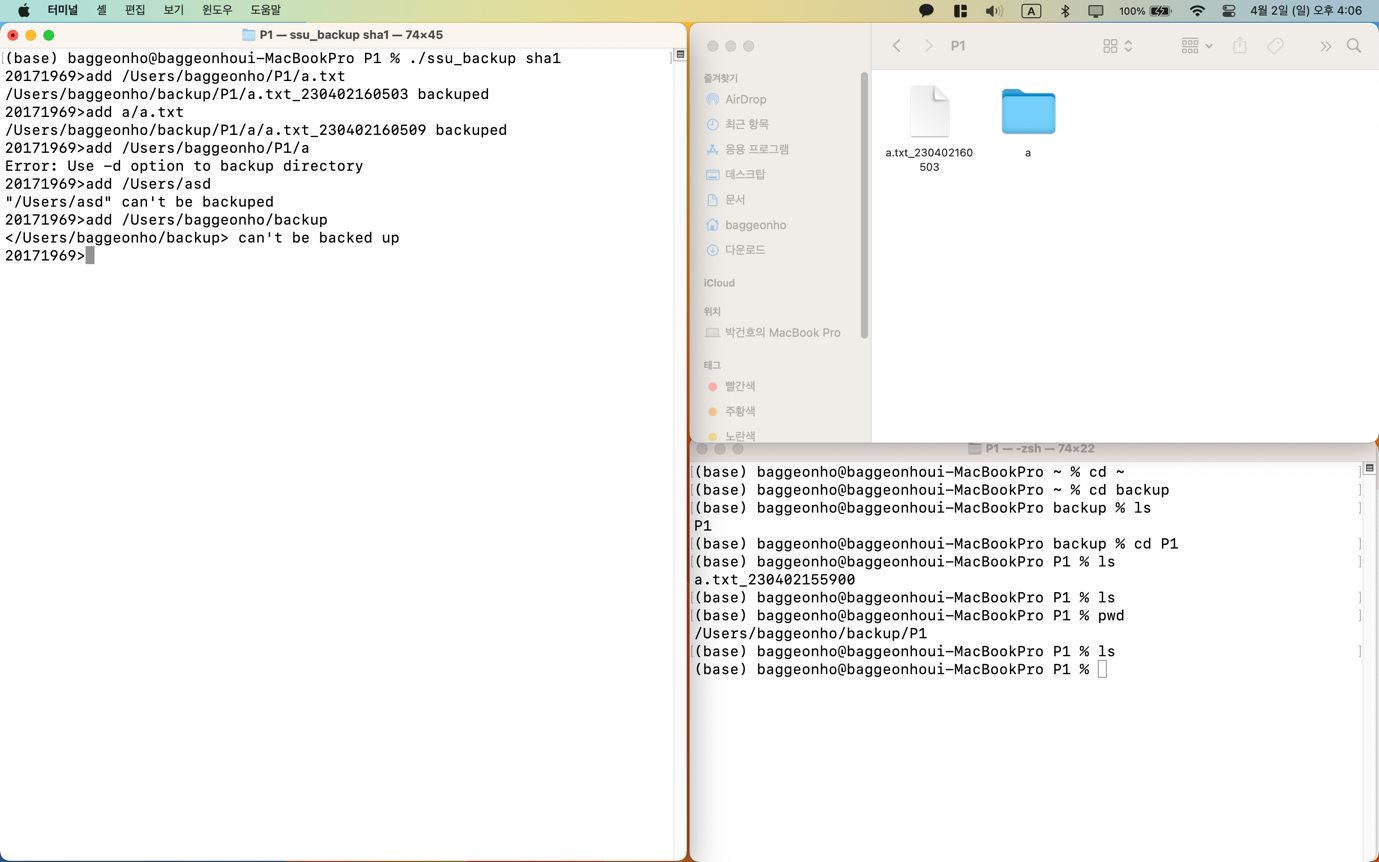
┃ ┃┗ c.txt

┃ ┗ c.txt

┣ a.txt

┗ b/

┗ backup

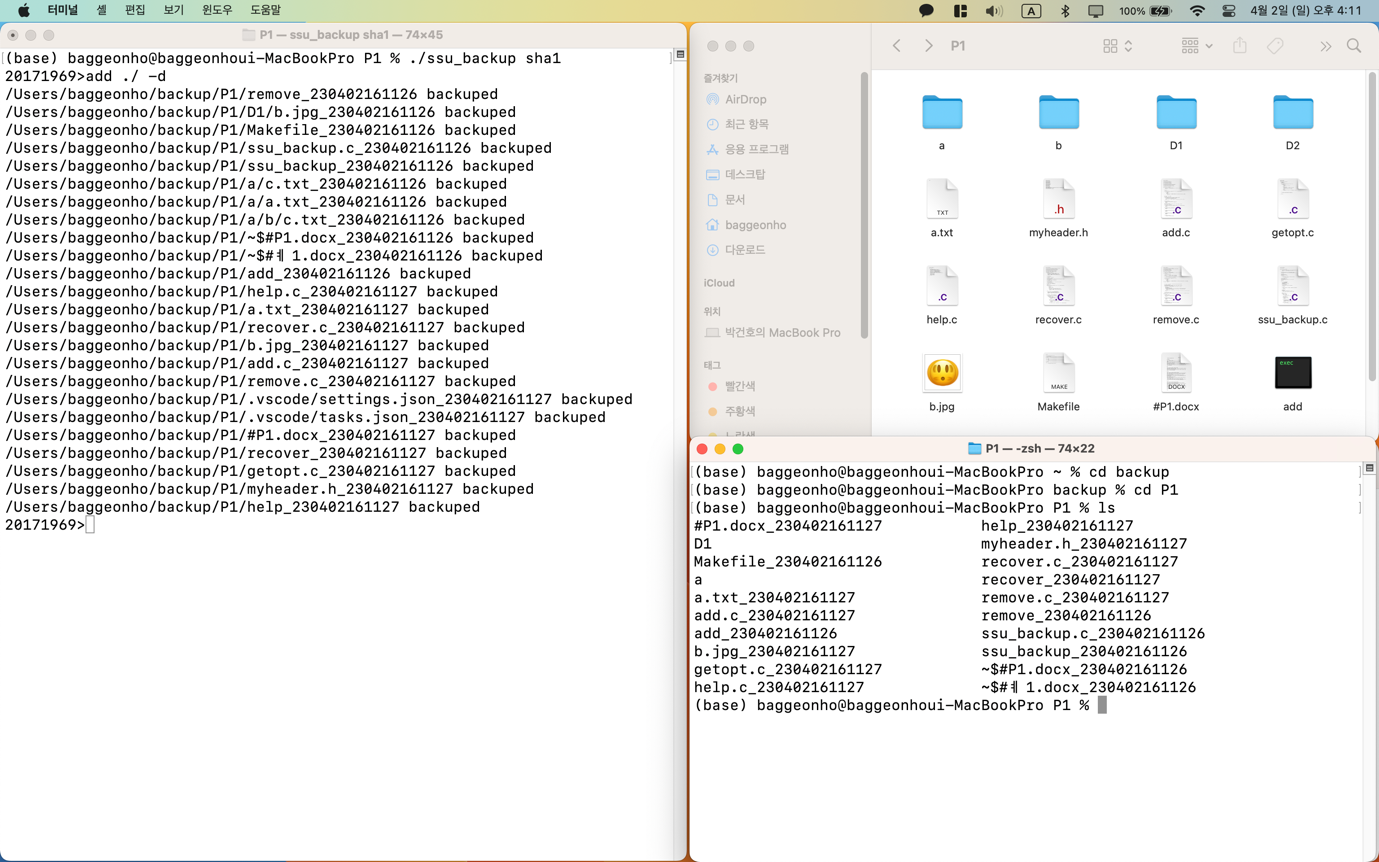


절대경로 입력 가능

디렉토리 옵션 -d없이 실행시 에러 출력

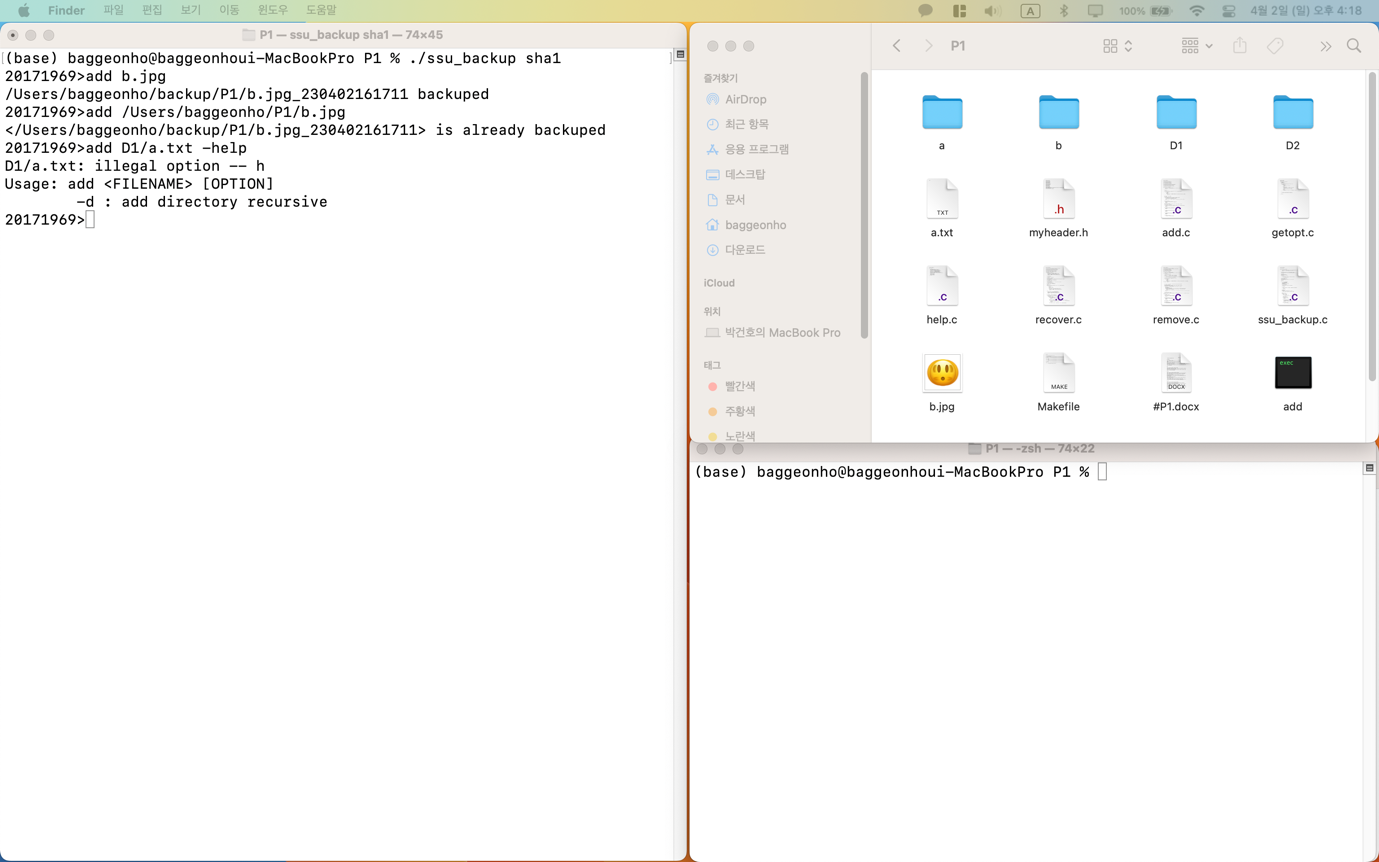
홈 디렉토리 벗어나서 에러 출력

백업 디렉토리는 백업하지 못하는 에러 출력



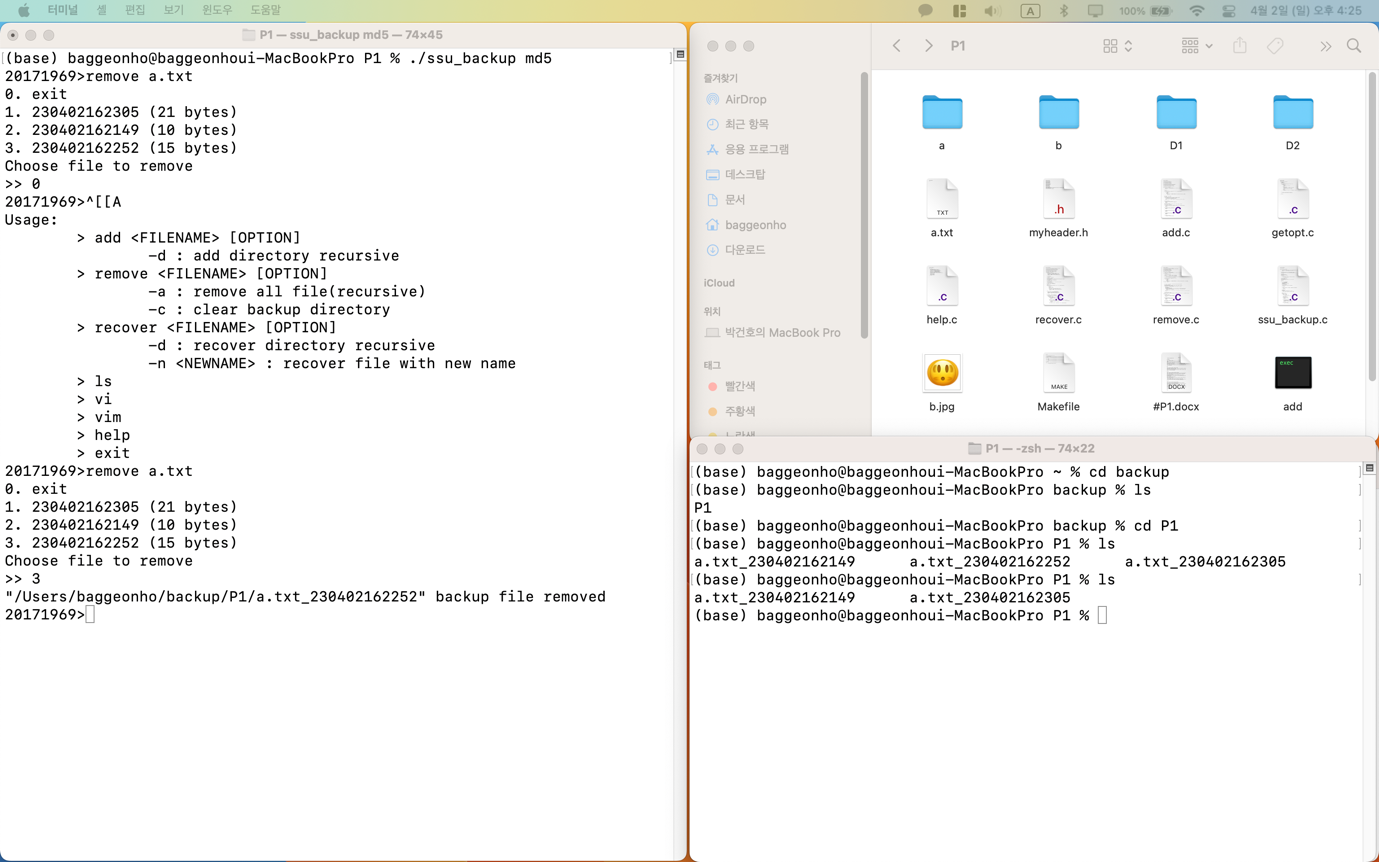
명세 그림2와같이

D2디렉토리에는 파일이 없어서 디렉토리가 백업 안된 것을 알 수 있음



완전히 동일한 파일에 대해서 해시값이 같아 백업이 안된 것을 알 수 있음

알맞지 않은 옵션값에 대해 에러처리가 됨을 알 수 있음



a.txt를 삭제하고자 한다.

vi에디터로 살짝의 수정을 해가며 3번의 백업을 함.

0번을 고를때 삭제가 안되고 3번을 골랐을때 3번이 삭제됨.

텍스트이(가) 표시된 사진

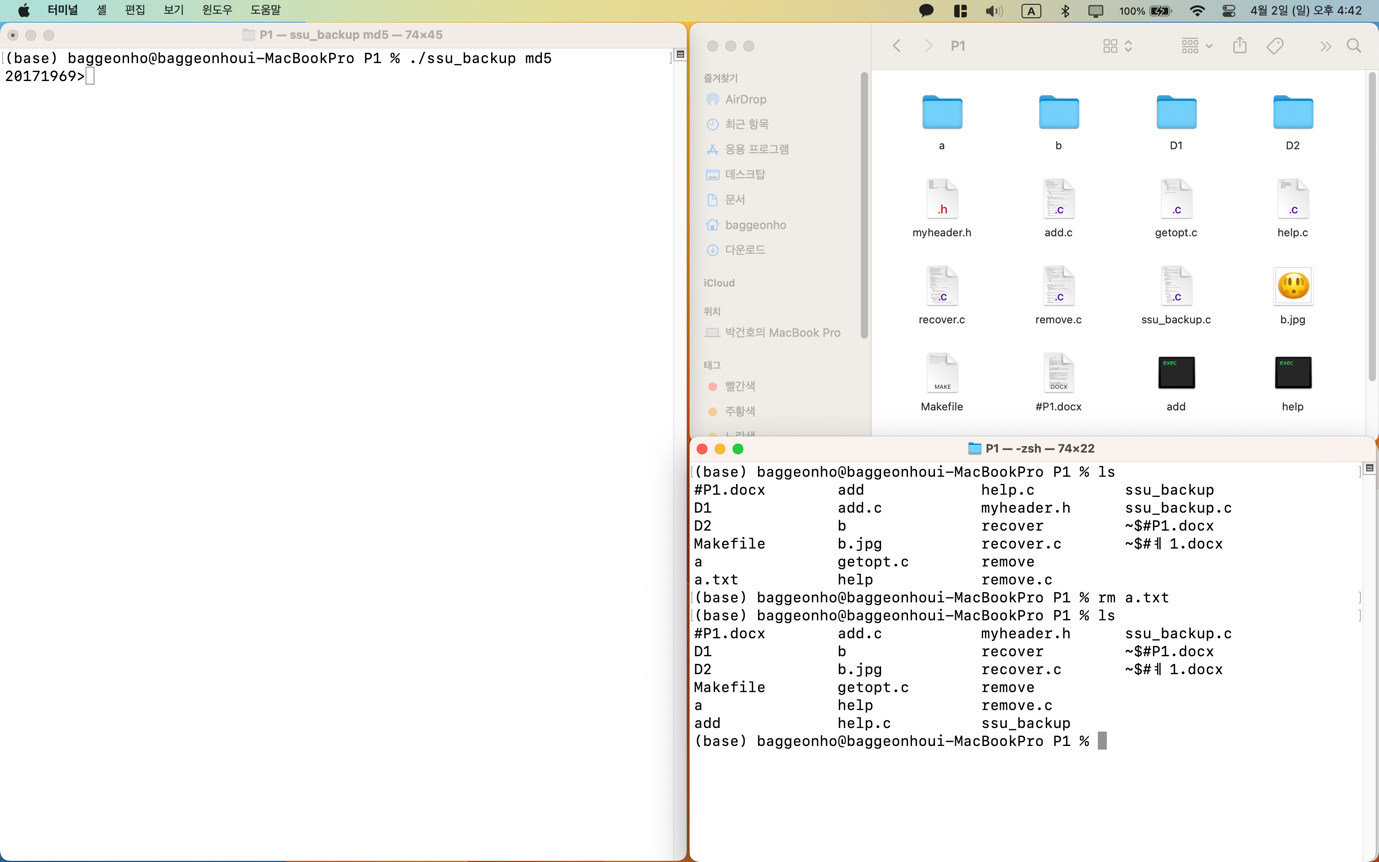
자동 생성된 설명

-a 옵션 사용 가능

잘못된 옵션 사용시 에러 출력

remove -c 명령시에 backup 디렉토리 내 모든 파일 삭제 후 삭제한 파일과 디렉토리 개수 출력

남아있는 파일이 없어서 예외처리 7번 출력



현재 P1디렉토리에는 a.txt가 없음



리커버리 확인



/Users/baggeonho/backup/P1에 있는 a.txt 백업파일 두개

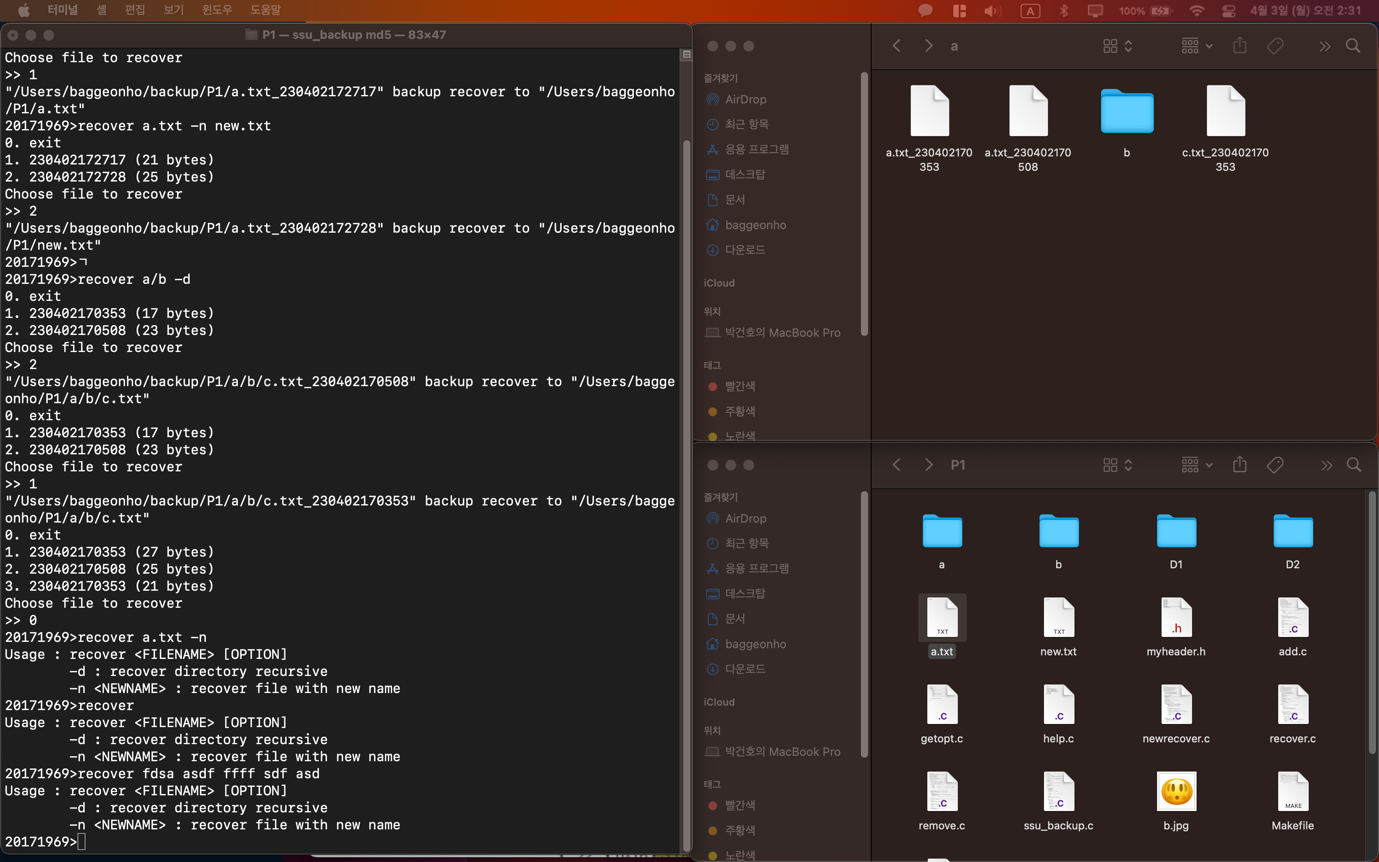
a.txt 입력으로 리커버리 요청시 두개 중 선택 가능

-n 옵션으로 새로운 이름 입력시 새로운 이름으로 저장

웹사이트이(가) 표시된 사진

자동 생성된 설명

-d 옵션 사용시 해당 디렉토리 하에 있는 파일 및 디렉터리를 재귀적으로 저장

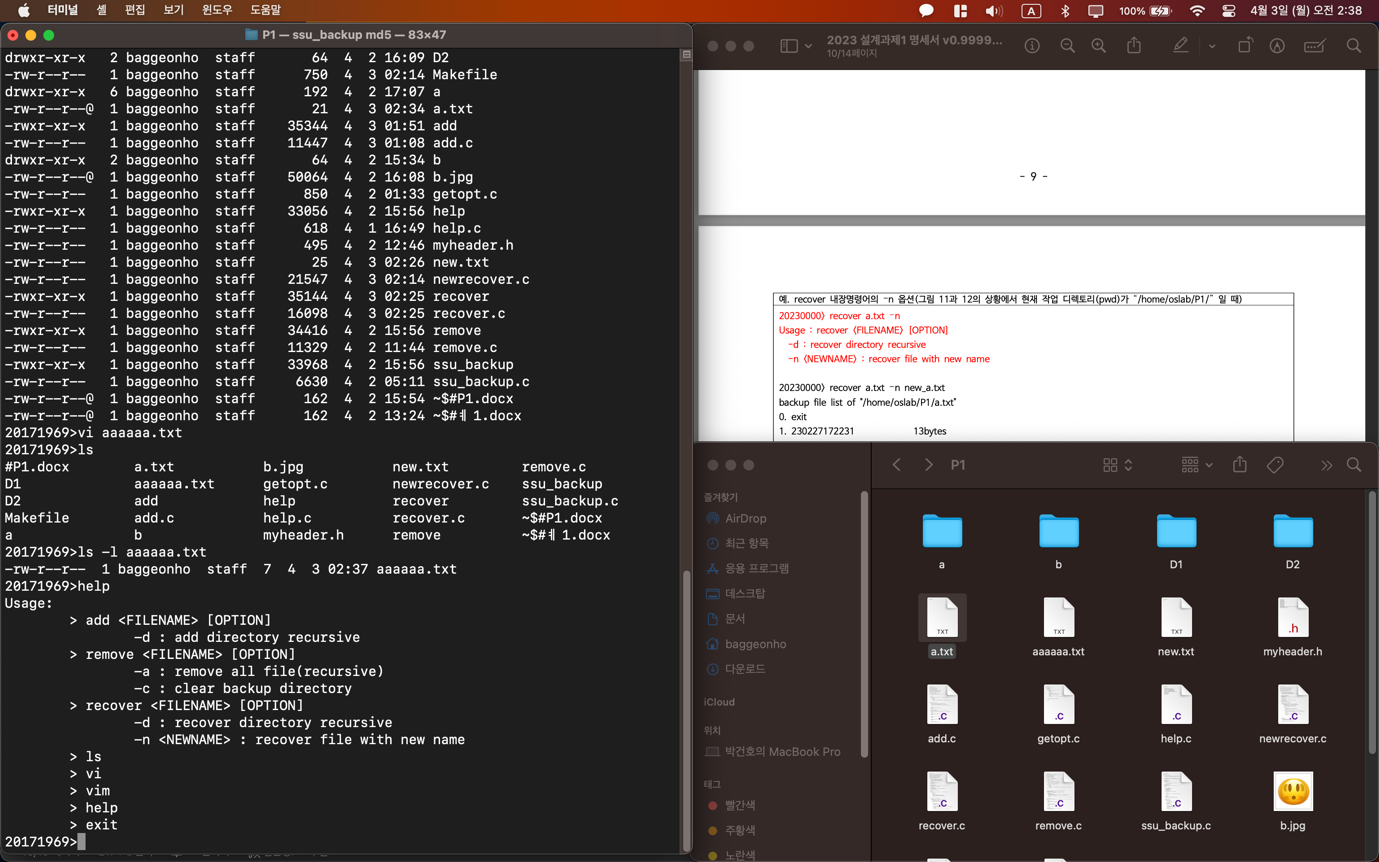


오류 출력 화면

텍스트, 블랙, 전자제품이(가) 표시된 사진

자동 생성된 설명

ls 명령어 실행



vi 명령어로 aaaaaa.txt 생성 후 ls 명령어로 정보 확인

성공적으로 저장되었음을 알 수 있음

help명령어 실행 성공

5. 비고

openssl 버전이 deprecated 되었다는 오류 - MD5가 더이상 쓰이지 않는 안정성 떨어지는 암호화 함수 - 를 해결하기 위해 헤더파일에 openssl\_api\_compat 포함하여 MD5 사용.

본인의 환경에서 컴파일시에 openssl 경로를 찾지 못해 컴파일이 안되는 오류가 있었기에 Makefiledp 경로 옵션을 추가하여 라이브러리를 인클루드해서 컴파일 하였음

MAC OS 특성상 디렉토리에 .DS\_Store 라는 파인더 GUI 용 파일이 있을 수 있는데, 과제 구현상으로 같이 백업되거나 삭제되는 경우가 있어 "." 과 ".."에 더불어 이 파일을 포함하지 않는다는 구현상의 수정이 있었음

리눅스 환경과 다르게 BSD기반 맥OS는 st\_size 의 형식 지정자를 %lld를 사용하여 %lld 로 사용하였음