

<b>Course</b>	<b>고급시스템 프로그래밍</b>
<b>Instructor report ID</b>	<b>왕선태 Project 4 [ Command ]</b>
<b>Due date</b>	<b>2014.11.05</b>

<b>Department</b>	Computer Engineering
<b>Student id.</b>	20093284
<b>Student name</b>	나용철
<b>Submission date</b>	2014.11.05





# Command 목차

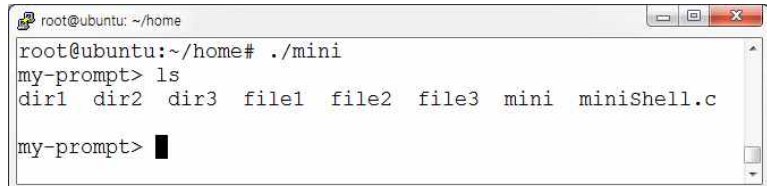

ls	4
od	8
cat	11
touch	14
head/tail	17
chmod	20
cd	23
pwd	24
cp	25
mv	29
rm	32
ln	34
mkdir	37
who	39
find	40
별첨 [Souce code]	43

# \* Command 구현 Manual

-참고 : od, cat, head, tail명령어는 Input File이 없다면 Standar Input으로 입력을 받는다.

## <매뉴얼 Format>

<명령어>	<구현설명>	<설명>	
	<테스트결과>	<Command>	<설명>
		<명령어 실행 >  [ linux 실제 수행 장면 ] 	
		[ 직접 구현한 커맨드 수행 장면] 	
		<옵션>	<설명>
	<명령어 옵션 실행 >  [ linux 실제 수행 장면 ]   [ 직접 구현한 커맨드 수행 장면] 		
사용함수	<사용함수들 설명>		

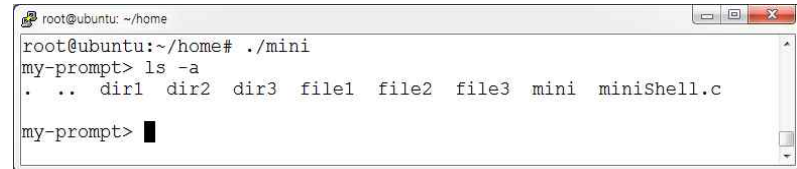
ls	<div>&lt;구현설명&gt;</div> <div>현재 디렉토리를 기준으로 모든 파일과 디렉토리를 출력가능하게 하는 명령어다. 또한 입력받은 디렉토리를 검색할 수 도 있다. 만약 입력받은 인자 디렉토리가 없다면 현재의 위치를 탐색한다. 입력받은 인자의 옵션을 검사하여 flag를 설정하여 준다. 그 후 현재 디렉토리 경로를 탐색하기위해 EntrySearch()함수를 수행한다. 이 함수는 입력받은 디렉토리를 열고 내부 엔트리들의 이름을 모두 2차배열에 스트링들로 저장을 한 후 해당 리스트를 출력해주는 함수 EntryPrint() 에 인자로 전달한다. 혹은 'R'옵션이 설정되어 있는 경우, 리스트내의 파일중 폴더를 찾아내어 EntrySearch()함수를 recursive하게 호출하여 하위 디렉토리를 탐색 할 수 있도록 하였다. 출력함수인 EntryPrint()함수는 인자로 전달받은 엔트리 각각의 이름을 출력한다. 만약 'l'옵션이 선택되어 있다면 포맷을 맞추기위해 해당 파일의 Permission Mode, Link Counter, User Name, Group Name, Size, 수정시간 과 이름을 출력하도록 한다.<ul style="list-style-type: none"><li>- Permission Mode는 StrMode()라는 함수에 퍼미션 모드 숫자와 스트링을 인자로 전달한다. 그러면 퍼미션 숫자를 스트링화 시켜 전달하여주는 함수를 통해 전달받는다.</li><li>- Link Counter는 파일정보 구조체에 있다.</li><li>- User Name는 파일의 UID번호를 패스워드 파일과 대조하여 이름에 해당하는 정보를 가져온다.</li><li>- Group Name은 그룹이름정보를 가져오는 함수를 통해 전달 받는다.</li><li>- Size는 파일정보 구조체에 있다.</li><li>- 수정시간은 파일정보 구조체에 있는 시간을 해당 포맷에 맞추어 변형시켜서 저장한다.</li></ul></div>
	<div>&lt;테스트결과&gt;</div> <div><div>[ ls [옵션] [파일] ]</div><div>디렉터리 목록을 출력 한다</div><div><div>[ linux ]</div><div>root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3</div><div>[ 결과 ]</div><div></div><div></div></div></div>

-a	경로안의 모든 내용을 출력한다
----	------------------

[ linux ]

```
root@ubuntu:~/home# ls -a
.  ..  dir1  dir2  dir3  file1  file2  file3
```

[ 결과 ]

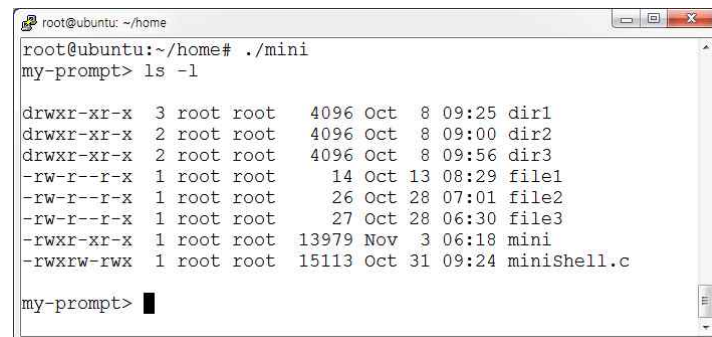


-l	긴 리스트의 포맷으로 출력한다
----	------------------

[ linux ]

```
root@ubuntu:~/home# ls -l
total 12
drwxr-xr-x 3 root root 4096 Oct  8 07:25 dir1
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir2
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir3
-rw-r--r-- 1 root root    0 Oct  8 07:21 file1
-rw-r--r-- 1 root root    0 Oct  8 07:21 file2
-rw-r--r-- 1 root root    0 Oct  8 07:22 file3
```

[ 결과 ]



-R	현재 디렉터리를 기준으로 모든 하위의 디렉터리를 출력한다.
----	----------------------------------

[ linux ]

```
root@ubuntu:~/home# ls -R
.:
dir1  dir2  dir3  file1  file2  file3

./dir1:
dir4  file4


./dir1/dir4:
dir5

./dir1/dir4/dir5:

./dir2:

./dir3:
```

[ 결과 ]



```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls -R
/root/home :
dir1 dir2 dir3 file1 file2 file3 mini minishell.c

/root/home/dir1 :
dir4 file4

/root/home/dir1/dir4 :
dir5

/root/home/dir1/dir4/dir5 :

/root/home/dir2 :

/root/home/dir3 :

my-prompt> █

```

-r

정렬의 순서를 역방향으로 한다

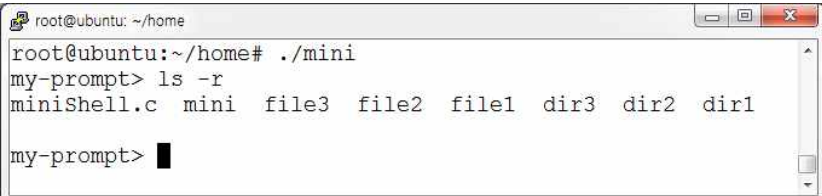
[ linux ]

```

root@ubuntu:~/home# ls -r
file3 file2 file1 _dir3 dir2 dir1

```

[ 결과 ]



```

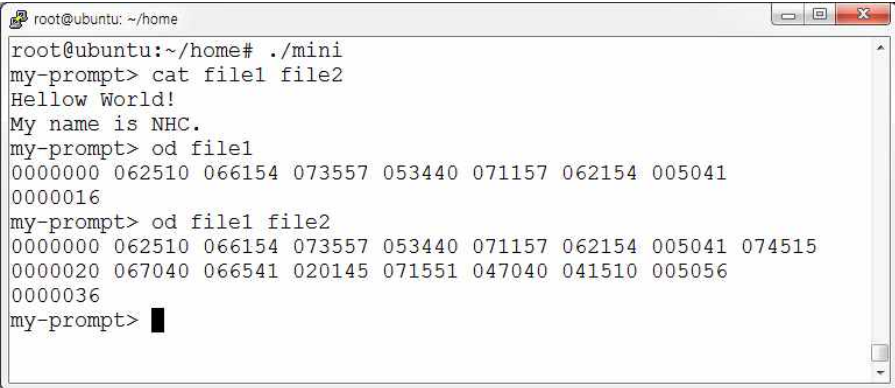
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls -r
minishell.c mini file3 file2 file1 dir3 dir2 dir1

my-prompt> █

```

사용함수

1. “unistd.h” char \*getcwd(char \*buf, size\_t size);
  - 기능 : 작업 디렉토리의 전체 이름을 구합니다.
  - 인수 : char \*buf → 작업 디렉토리 문자열을 담을 버퍼  
size\_t size → 버퍼의 크기
  - 반환값 : [성공] 현재 작업 디렉토리  
[실패] -1
2. “dirent.h” DIR \*opendir(const char \*name);
  - 기능 : 지정한 디렉토리 열기를 합니다. 디렉토리 안의 내용을 보기 위함 이다.
  - 인수 : char \*name → 열기 대상 디렉토리
  - 반환값 : [성공] 디렉토리 정보 구조체인 DIR 포인터  
[실패] NULL
3. “dirent.h” struct dirent \*readdir( DIR \*dir);
  - 기능 : 열기를 한 디렉토리 안에 있는 파일 중 첫번째 파일에 대한 정보를 구합니다.
  - 인수 : DIR \*dir → opendir()에서 열기한 디렉토리 정보
  - 반환값 : [성공] 파일이나 디렉토리 정보  
[실패] NULL
4. “sys/stat.h” int stat(const char \*pathname, struct \_stat \*buf);
  - 기능 : 파일 정보를 확인하는 함수
  - 인수 : const char \*pathname → 파일 또는 폴더 경로  
struct \_stat \*buf → 구조체 stat 의 포인터
  - 반환값 : [성공] 0, buf인자에 stat구조체에 파일  
[실패] -1
5. “pwd.h” struct passwd \*getpwuid(uid\_t uid);
  - 기능 : 유저 ID로 사용자 정보를 구합니다.
  - 인수 : uid\_t uid → 유저 아이디
  - 반환값 : [성공] 사용자 정보를 담고 있는 struct passwd 포인터  
[실패] NULL
6. “grp.h” struct group \*getgrgid(gid\_t gid);
  - 기능 : 그룹 아이디에 대해 그룹 파일로부터 그룹 정보를 구합니다.
  - 인수 : gid\_t gid → 그룹 ID
  - 반환 : [성공] 그룹정보  
[실패] NULL

	<구현설명>	<p>od명령어는 입력받은 파일들을 8진수 혹은 아스키로 변환해 출력해주는 명령어다. 만약 전달받은 입력 파일이 없다면 redirection된 것이므로 파일에서 읽어오지 않고 standard input에서 읽어온다. 입력받은 파일들은 open()함수 읽기 모드로 열고 read()함수로 파일의 내용을 모두 읽어 임시 파일을 open()함수 쓰기모드로 열고 write()함수를 사용해 옮겨 적는다. 옮겨진 이 임시 파일을 fopen()함수로 열고 f 출력되기 위해서는 fread()함수로 1바이트씩 읽혀진다. 이때 아스키모드 옵션이 추가 되어있다면 1바이트 그대로 출력을 하게 되고, 옵션이 설정되어 있지 않는다면 처음 1바이트를 저장해두고 다음 읽은 1바이트를 왼쪽으로 1바이트 쉬프트한 후 이전 바이트를 추가시켜서 총 2바이트를 출력한다. 한 줄에는 총 16바이트를 출력하며 라인 첫 번째에는 출력된 바이트 수와 함께 마지막 줄에는 출력된 총 바이트 수를 8진수로 나타낸다. 끝으로 임시파일을 unlink()함수로 제거 해준다.</p>	
od	<테스트결과>	<p>[ od [옵션] 파일.. ]</p>	<p>지정된 파일의 내용을 8진수로 dump하여 보여준다. 입력 파일은 여러개의 파일이 올 수 있다.</p>
		<p>[ linux ]</p> <pre> root@ubuntu:~/home# cat file1 file2 Hellow World! My name is NHC. root@ubuntu:~/home# od file1 0000000 062510 066154 073557 053440 071157 062154 005041 0000016 root@ubuntu:~/home# od file1 file2 0000000 062510 066154 073557 053440 071157 062154 005041 074515 0000020 067040 066541 020145 071551 047040 041510 005056 0000036 </pre> <p>[ 결과 ]</p>  <pre> root@ubuntu: ~/home root@ubuntu:~/home# ./mini my-prompt&gt; cat file1 file2 Hellow World! My name is NHC. my-prompt&gt; od file1 0000000 062510 066154 073557 053440 071157 062154 005041 0000016 my-prompt&gt; od file1 file2 0000000 062510 066154 073557 053440 071157 062154 005041 074515 0000020 067040 066541 020145 071551 047040 041510 005056 0000036 my-prompt&gt; █ </pre>	



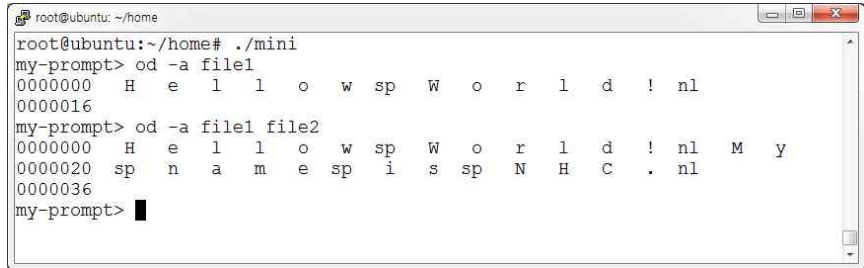
-a

파일의 내용을 1byte ASCII  
문자로 표현한다

#### [ linux ]

```
root@ubuntu:~/home# od -a file1
0000000 H e l l o w s p W o r l d ! nl
0000016
root@ubuntu:~/home# od -a file1 file2
0000000 H e l l o w s p W o r l d ! nl M y
0000020 sp n a m e sp i s sp N H C . nl
0000036
```

#### [ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> od -a file1
0000000 H e l l o w s p W o r l d ! nl
0000016
my-prompt> od -a file1 file2
0000000 H e l l o w s p W o r l d ! nl M y
0000020 sp n a m e sp i s sp N H C . nl
0000036
my-prompt>
```

#### 사용 함수

1. `int open (const char *FILENAME, int FLAGS[, mode_t MODE])`
  - 헤더 : "fcntl.h"
  - 기능 : 리눅스에서 제공하는 함수로 파일을 사용하기 위해 여러 모드로 열기한다.
  - 인수 : `char *FILENAME` → 대상 파일 이름  
`int FLAGS` → 파일에 대한 열기 옵션  
`[, mode_t MODE]` → `O_CREAT` 옵션 사용에 의해 파일이 생성될 때 지정되는 파일 접근 권한
  - 반환값 : [성공] 일 디스크립터의 양의 정수 값  
[실패] -1

옵션	설명
<code>RDONLY</code>	읽기 전용으로 열기
<code>O_WRONLY</code>	쓰기 전용으로 열기
<code>O_RDWR</code>	읽기와 쓰기가 모두 가능
<code>_CREAT</code>	해당 파일이 없으면 생성합니다. <code>O_CREAT</code> 로 파일을 생성하게 된다면 파일의 접근권한을 지정하기 위해 접근 권한 값을 추가해야 합니다.
<code>O_EXCL</code>	<code>O_CREAT</code> 를 사용할 때, <code>O_EXCL</code> 를 함께 사용하면, 이미 파일이 있을 때에는 <code>open()</code> 되지 않아 이전 파일을 보존할 수 있습니다.
<code>O_TRUNC</code>	기존의 파일 내용을 모두 삭제합니다.
<code>O_APPEND</code>	파일을 추가하여 쓰기가 되도록 <code>open</code> 후에 쓰기 포인터가 파일의 끝에 위치하게 됩니다.
<code>O_NOCTTY</code>	열기 대상이 터미널일 경우, 이 터미널이 프로그램의 제어 터미널로 할당하지 않습니다.
<code>O_NONBLOCK</code>	읽을 내용이 없을 때에는 읽을 내용이 있을 때까지 기다리지 않고 바로 복귀합니다.
<code>O_SYNC</code>	쓰기를 할 때, 실제 쓰기가 완료될 때 까지 기다립니다. 즉, 물리적으로 쓰기가 완료되어야 복귀하게 됩니다.

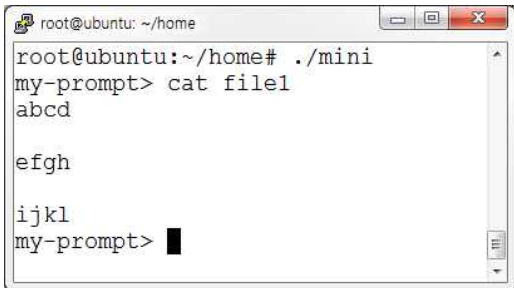
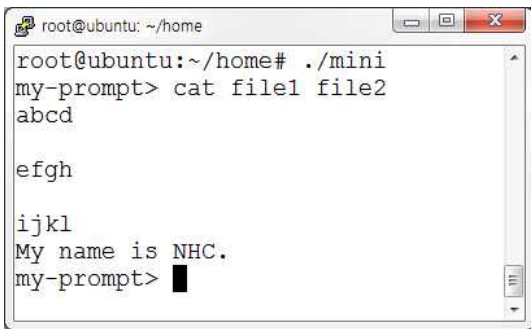
2. `ssize_t read (int fd, void *buf, size_t nbytes)`
  - 헤더 : "unistd.h"
  - 기능 : `open()` 함수로 열기를 한 파일의 내용을 읽기를 한다.
  - 인수 : `int fd` → 파일 디스크립터  
`void *buf` → 파일을 읽어 들일 버퍼  
`size_t nbytes` → 버퍼의 크기
  - 반환값 : [성공] 읽어들이는 바이트 수

[실패] -1

3. ssize\_t write (int fd, const void \*buf, size\_t n)
  - 헤더 : "unistd.h"
  - 기능 : open() 함수로 열기를 한 파일에 내용을 쓰기 한다.
  - 인수 : int fd → 파일 디스크립터  
 void \*buf → 파일에 쓰기를 할 내용을 담은 버퍼  
 size\_t n → 쓰기할 바이트 수
  - 반환값 : [성공] 쓰기 한 바이트 수  
 [실패] -1
  
4. int close (int fd)
  - 헤더 : "unistd.h"
  - 기능 : open() 함수로 열기를 한 파일의 사용을 중지한다.
  - 인수 : int fd → 파일 디스크립터
  - 반환값 : [성공] 0  
 [실패] -1
  
5. int fopen (const char \*path, struct stat \*buf);
  - 헤더 : "stdio.h"
  - 기능 : 파일을 읽어 들이거나 쓰기를 하기 위해 파일을 여는 함수
  - 인수 : char \*path → 열고자하는 파일의 전체 이름  
 char \*mode → 파일 열기를 위한 옵션, 사용 형태

모드	설명
r	읽기 전용. 파일이 있어야 한다.
w	쓰기 전용. 파일이 없으면 생성되고, 파일이 있다면 기존 내용은 지운다
a	내용 추가. 파일이 없으면 생성되고, 파일이 있다면 기존 내용 뒤에 추가 된다
rb	바이너리 파일 읽기 전용으로 열기
wb	바이너리 파일 쓰기 전용으로 열기
ab	바이너리 파일 추가용으로 열기
r+	읽기와 쓰기용으로 열기
w+	읽기와 쓰기용으로 열기

- 반환값 : [성공] 파일 포인터  
 [실패] NULL
- 
6. size\_t fread(void \*ptr, size\_t size, size\_t nitems, FILE \*stream);
    - 헤더 : "stdio.h"
    - 기능 : 파일을 통해 데이터를 읽어 들인다.
    - 인수 : void \*ptr → 파일 내용을 읽어 들일 메모리 포인터  
 size\_t size → 데이터 하나의 크기  
 size\_t nitems → 읽어 들일 데이터의 개수  
 FILE \*stream → 대상 파일 스트림
    - 반환값 : [성공] 읽어들이는 데이터 개수  
 [실패] -1
  
  7. int unlink(const char \*path)
    - 헤더 : "unistd.h"
    - 기능 : path지정된 파일과 디렉토리의 inode에서 링크 수를 감소시킨다.  
 링크수가 0이되면 path에 지정된 파일이 삭제된다.
    - 인수 : const char \*path → 삭제할 파일의 경로
    - 반환값 : [성공] 0  
 [실패] -1

	<p>&lt;구현설명&gt;</p>	<p>cat 명령어는 입력받은 파일들을 출력하는 명령어다. 우선 넘겨받은 인자들 중 옵션이 존재하는 가를 체크하고 파일의 유무를 확인하여 redirection된 있는 가를 점검 한다. 만약 전달받은 입력 파일이 없다면 redirection된 것이므로 파일에서 읽어오지 않고 standard input에서 읽어온다. open()함수로 파일을 열고 read()함수로 파일의 내용을 1바이트씩 읽으며 스트링변수에 읽은 캐릭터를 쌓아둔다. 또한 ‘\n’인가를 검사하면서 new line을 만나면 모아두었던 캐릭터들의 스트링을 화면에 출력한다. 이는 입력받은 파일의 한줄과 같다. 옵션이 설정 되어 있는 경우 한줄한줄 읽을 때마다 카운트를 해주면서 다음 스트링에 카운트를 붙여주면 된다. 그러나 b옵션의 경우는 공백을 제외하므로 한줄에 ‘\n’만 있다면 그냥 ‘\n’만 출력을 한다.</p>	
cat		[ cat [옵션] [파일].. ]	텍스트 파일내용을 출력 한다. 여러 입력 파일들이 올 수 있다.
	<테스트결과>	<p>[ linux ]</p> <pre>root@ubuntu:~/home# cat file1 abcd  efgh  ijkl  root@ubuntu:~/home# cat file1 file2 abcd efgh ijkl My name is NHC. root@ubuntu:~/home#</pre> <p>[ 결과 ]</p>  	

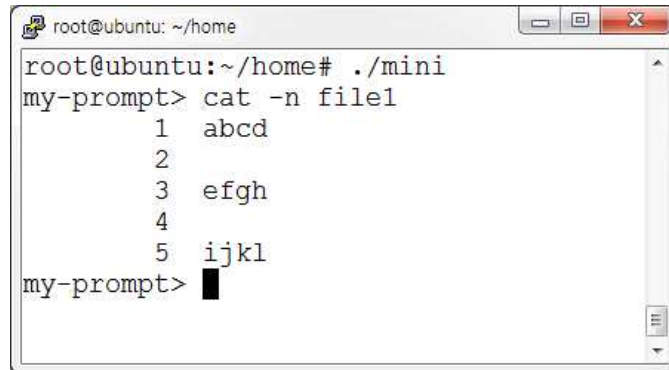
-n

각 문장 앞에 번호를 표시해준다

[ linux ]

```
root@ubuntu:~/home# cat -n file1
 1 abcd
 2
 3     efgh
 4
 5 ijk1
```

[ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> cat -n file1
 1 abcd
 2
 3 efgh
 4
 5 ijk1
my-prompt> █
```

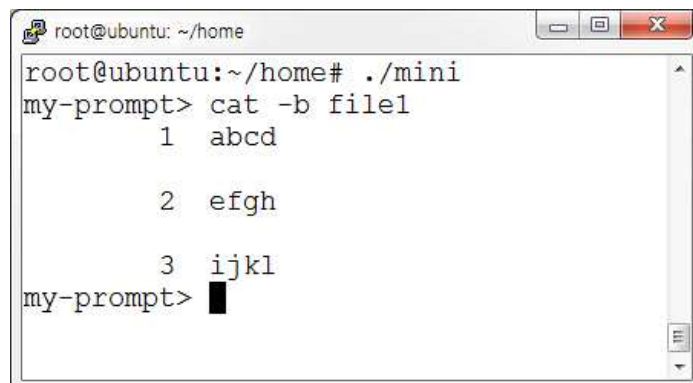
-b

각 문장 앞에 번호를 표시해준다  
(공백제외)

[ linux ]

```
root@ubuntu:~/home# cat -b file1
 1 abcd
    2     efgh
    3 ijk1
```

[ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> cat -b file1
 1 abcd
    2 efgh
    3 ijk1
my-prompt> █
```

## 사용함수

1. `int open (const char *FILENAME, int FLAGS[, mode_t MODE])`
  - 헤더 : "fcntl.h"
  - 기능 : 리눅스에서 제공하는 함수로 파일을 사용하기 위해 여러 모드로 열기한다.
  - 인수 : `char *FILENAME` → 대상 파일 이름  
`int FLAGS` → 파일에 대한 열기 옵션  
`[, mode_t MODE]` → `O_CREAT` 옵션 사용에 의해 파일이 생성될 때 지정되는 파일 접근 권한
  - 반환값 : [성공] 일 디스크립터의 양의 정수 값  
[실패] -1

옵션	설명
<code>RDONLY</code>	읽기 전용으로 열기
<code>O_WRONLY</code>	쓰기 전용으로 열기
<code>O_RDWR</code>	읽기와 쓰기가 모두 가능
<code>_CREAT</code>	해당 파일이 없으면 생성합니다. <code>O_CREAT</code> 로 파일을 생성하게 된다면 파일의 접근권한을 지정하기 위해 접근 권한 값을 추가해야 합니다.
<code>O_EXCL</code>	<code>O_CREAT</code> 를 사용할 때, <code>O_EXCL</code> 를 함께 사용하면, 이미 파일이 있을 때에는 <code>open()</code> 되지 않아 이전 파일을 보존할 수 있습니다.
<code>O_TRUNC</code>	기존의 파일 내용을 모두 삭제합니다.
<code>O_APPEND</code>	파일을 추가하여 쓰기가 되도록 <code>open</code> 후에 쓰기 포인터가 파일의 끝에 위치하게 됩니다.
<code>O_NOCITYTY</code>	열기 대상이 터미널일 경우, 이 터미널이 프로그램의 제어 터미널로 할당하지 않습니다.
<code>O_NONBLOCK</code>	읽을 내용이 없을 때에는 읽을 내용이 있을 때까지 기다리지 않고 바로 복귀합니다.
<code>O_SYNC</code>	쓰기를 할 때, 실제 쓰기가 완료될 때 까지 기다립니다. 즉, 물리적으로 쓰기가 완료되어야 복귀하게 됩니다.

2. `ssize_t read (int fd, void *buf, size_t nbytes)`
  - 헤더 : "unistd.h"
  - 기능 : `open()` 함수로 열기를 한 파일의 내용을 읽기를 한다.
  - 인수 : `int fd` → 파일 디스크립터  
`void *buf` → 파일을 읽어 들일 버퍼  
`size_t nbytes` → 버퍼의 크기
  - 반환값 : [성공] 읽어들이는 바이트 수  
[실패] -1
3. `ssize_t write (int fd, const void *buf, size_t n)`
  - 헤더 : "unistd.h"
  - 기능 : `open()` 함수로 열기를 한 파일에 내용을 쓰기 한다.
  - 인수 : `int fd` → 파일 디스크립터  
`void *buf` → 파일에 쓰기를 할 내용을 담은 버퍼  
`size_t n` → 쓰기할 바이트 수
  - 반환값 : [성공] 쓰기 한 바이트 수  
[실패] -1
4. `int close (int fd)`
  - 헤더 : "unistd.h"
  - 기능 : `open()` 함수로 열기를 한 파일의 사용을 중지한다.
  - 인수 : `int fd` → 파일 디스크립터
  - 반환값 : [성공] 0  
[실패] -1

touch	<구현설명>	<p>touch명령어는 open()함수로 파일이 존재한가 존재하지 않은가를 검사하고 존재하지 않는다면 생성한 후 변경시간을 변경하며, 존재한다면 그냥 해당 기존파일의 변경시간을 변경하는 명령어다. t옵션이 주어진다면 주어진 시간을 utimbuf 구조체에 변경시간과 접근시간을 입력하여 utime()함수로 변경된 utimbuf 구조체 값으로 시간을 변경한다.</p>	
	<테스트결과>	<p>[ touch [옵션] 파일명 ]</p>	<p>빈 파일을 생성하거나 기존 파일의 시간을 변경한다. 여러 파일들을 생성할 수 있다.</p>
<div data-bbox="555 607 663 636" data-label="Section-Header">[ linux ]</div> <div data-bbox="555 651 1342 864" data-label="Text"> <pre> root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 root@ubuntu:~/home# touch file4 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file4 root@ubuntu:~/home# touch file5 file6 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file4 file5 file6 </pre> </div> <div data-bbox="555 920 652 949" data-label="Section-Header">[ 결과 ]</div> <div data-bbox="555 965 1353 1240" data-label="Image"> <p>A terminal window titled 'root@ubuntu: ~/home' showing the execution of a mini shell. The user runs './mini' to start the shell. Inside the shell, the user runs 'ls' showing 'dir1 dir2 dir3 file1 file2 file3 mini miniShell.c'. Then the user runs 'touch file4'. After running 'ls' again, the output is 'dir1 dir2 dir3 file1 file2 file3 file4 mini miniShell.c', confirming that file4 was created.</p> </div> <div data-bbox="555 1256 1353 1518" data-label="Image"> <p>A terminal window titled 'root@ubuntu: ~/home' showing the execution of a mini shell. The user runs './mini' to start the shell. Inside the shell, the user runs 'ls' showing 'dir1 dir2 dir3 file1 file2 file3 file4 mini miniShell.c'. Then the user runs 'touch file5 file6'. After running 'ls' again, the output is 'dir1 dir2 dir3 file1 file2 file3 file4 file5 file6 mini miniShell.c', confirming that files5 and file6 were created.</p> </div>			

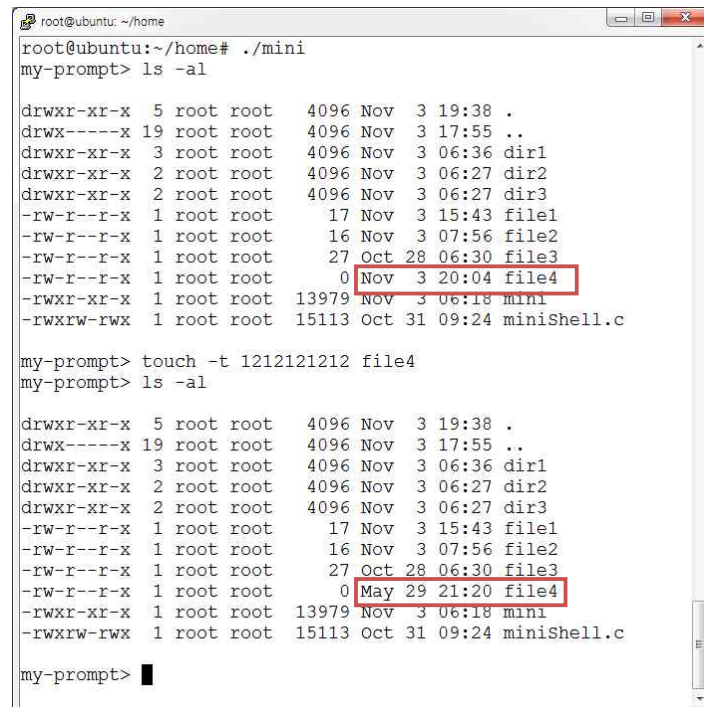
-t

현재시간 대신 지정한 시간으로  
변경한다 [YYMMDDHH/MM]

### [ linux ]

```
root@ubuntu:~/home# ls -l
total 16
drwxr-xr-x 3 root root 4096 Oct  8 07:25 dir1
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir2
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir3
-rw-r--r-- 1 root root  21 Oct  8 07:36 file1
-rw-r--r-- 1 root root   0 Oct  8 07:21 file2
-rw-r--r-- 1 root root   0 Oct  8 07:22 file3
-rw-r--r-- 1 root root   0 Oct  8 07:39 file4
root@ubuntu:~/home# touch -t 1212121212 file4
root@ubuntu:~/home# ls -l
total 16
drwxr-xr-x 3 root root 4096 Oct  8 07:25 dir1
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir2
drwxr-xr-x 2 root root 4096 Oct  8 07:21 dir3
-rw-r--r-- 1 root root  21 Oct  8 07:36 file1
-rw-r--r-- 1 root root   0 Oct  8 07:21 file2
-rw-r--r-- 1 root root   0 Oct  8 07:22 file3
-rw-r--r-- 1 root root   0 Dec 12 2012 file4
```

### [ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls -al

drwxr-xr-x  5 root root    4096 Nov  3 19:38 .
drwx-----x 19 root root    4096 Nov  3 17:55 ..
drwxr-xr-x  3 root root    4096 Nov  3 06:36 dir1
drwxr-xr-x  2 root root    4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root    4096 Nov  3 06:27 dir3
-rw-r--r-x  1 root root      17 Nov  3 15:43 file1
-rw-r--r-x  1 root root      16 Nov  3 07:56 file2
-rw-r--r-x  1 root root      27 Oct 28 06:30 file3
-rw-r--r-x  1 root root       0 Nov  3 20:04 file4
-rwxr-xr-x  1 root root   13979 Nov  3 06:18 mini
-rwxrw-rwx  1 root root   15113 Oct 31 09:24 miniShell.c

my-prompt> touch -t 1212121212 file4
my-prompt> ls -al

drwxr-xr-x  5 root root    4096 Nov  3 19:38 .
drwx-----x 19 root root    4096 Nov  3 17:55 ..
drwxr-xr-x  3 root root    4096 Nov  3 06:36 dir1
drwxr-xr-x  2 root root    4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root    4096 Nov  3 06:27 dir3
-rw-r--r-x  1 root root      17 Nov  3 15:43 file1
-rw-r--r-x  1 root root      16 Nov  3 07:56 file2
-rw-r--r-x  1 root root      27 Oct 28 06:30 file3
-rw-r--r-x  1 root root       0 May 29 21:20 file4
-rwxr-xr-x  1 root root   13979 Nov  3 06:18 mini
-rwxrw-rwx  1 root root   15113 Oct 31 09:24 miniShell.c

my-prompt> █
```

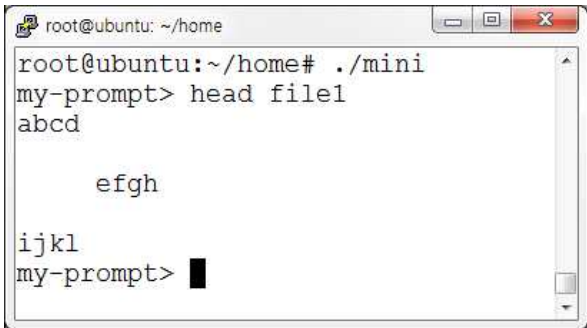
## 사용함수

1. `int open (const char *FILENAME, int FLAGS[, mode_t MODE])`
  - 헤더 : "fcntl.h"
  - 기능 : 리눅스에서 제공하는 함수로 파일을 사용하기 위해 여러 모드로 열기한다.
  - 인수 : `char *FILENAME` → 대상 파일 이름  
`int FLAGS` → 파일에 대한 열기 옵션  
`[, mode_t MODE]` → `O_CREAT` 옵션 사용에 의해 파일이 생성될 때 지정되는 파일 접근 권한
  - 반환값 : [성공] 일 디스크립터의 양의 정수 값  
[실패] -1

옵션	설명
<code>RDONLY</code>	읽기 전용으로 열기
<code>O_WRONLY</code>	쓰기 전용으로 열기
<code>O_RDWR</code>	읽기와 쓰기가 모두 가능
<code>O_CREAT</code>	해당 파일이 없으면 생성합니다. <code>O_CREAT</code> 로 파일을 생성하게 된다면 파일의 접근권한을 지정하기 위해 접근 권한 값을 추가해야 합니다.
<code>O_EXCL</code>	<code>O_CREAT</code> 를 사용할 때, <code>O_EXCL</code> 를 함께 사용하면, 이미 파일이 있을 때에는 <code>open()</code> 되지 않아 이전 파일을 보존할 수 있습니다.
<code>O_TRUNC</code>	기존의 파일 내용을 모두 삭제합니다.
<code>O_APPEND</code>	파일을 추가하여 쓰기가 되도록 <code>open</code> 후에 쓰기 포인터가 파일의 끝에 위치하게 됩니다.
<code>O_NOCTTY</code>	열기 대상이 터미널일 경우, 이 터미널이 프로그램의 제어 터미널로 할당하지 않습니다.
<code>O_NONBLOCK</code>	읽을 내용이 없을 때에는 읽을 내용이 있을 때까지 기다리지 않고 바로 복귀합니다.
<code>O_SYNC</code>	쓰기를 할 때, 실제 쓰기가 완료될 때 까지 기다립니다. 즉, 물리적으로 쓰기가 완료되어야 복귀하게 됩니다.

2. `int close (int fd)`
  - 헤더 : "unistd.h"
  - 기능 : `open()` 함수로 열기를 한 파일의 사용을 중지한다.
  - 인수 : `int fd` → 파일 디스크립터
  - 반환값 : [성공] 0  
[실패] -1
3. `int utime(const char *filename, struct utimbuf *buf);`
  - 헤더 : "unistd.h"
  - 기능 : `inode` 에 대한 접근/수정 시간을 변경한다.
  - 인수 : `time_t actime` → 접근시간  
`time_t modtime` → 변경시간
  - 반환값 : [성공] 0  
[실패] -1



head/tail	<구현설명>	<p>파이썬으로 구현했으며 head는 출력을 파일의 앞부분부터 진행하며, tail은 출력을 파일의 뒷부분부터 진행한다. 'n' 옵션이 주어지지 않으면 모든 출력물은 같은 출력물을 나타낸다.</p> <p>우선 옵션의 존재 여부를 확인한다. 만약 전달받은 입력 파일이 없다면 redirection된 것이므로 파일에서 읽어오지 않고 standard input에서 읽어온다. sys.stdin.readlines() 함수로 standard input에서 읽어온다. 전달받은 파일들이 있다면 시작부분부터 마지막 전달받은 파일들의 존재를 os.path.exists() 함수로 확인하고 존재 한다면 open() 함수로 파일을 열고 모든 내용들을 한번에 readlines() 함수로 모든 라인들을 한번에 스트링 배열에 저장한다. 저장된 스트링들은 한줄씩 출력하게 된다. 이때 '\n'은 제거된 상태에서 출력된다. 파이썬의 print 자체에 '\n'이 포함되기 때문이다. 이때 'n' 옵션이 설정되어 있다면 입력받은 숫자만큼의 출력이 시작되는 지점에서 출력된다. head는 처음부분부터 지정된 숫자만큼의 라인만큼 출력하며, tail은 끝부분부터 지정된 숫자만큼의 라인만큼 출력한다. 'v' 옵션이 설정되어 있다면 출력전에 파일 이름을, 'q' 옵션이라면 출력전에 파일 이름을 출력하지 않는다. (아무것도 없는 옵션이라면 기본값은 'q'이다.)</p>	
		[ head [옵션] 파일명 ] [ tail [옵션] 파일명 ]	파일의 첫 번째 부분을 출력 한다 파일의 마지막 부분을 출력 한다 두 명령어는 같은 출력결과를 내나 'n' 옵션 시 출력내용은 다르다.
	<테스트결과>	<p>[ linux ]</p> <pre>root@ubuntu:~/home# head file1 abcd  efgh  ijkl</pre> <p>[ 결과 ]</p> 	

-v

출력하는 파일명을 함께 출력한다

[ linux ]

```
root@ubuntu:~/home# head -v file1
==> file1 <==
abcd

    efgh

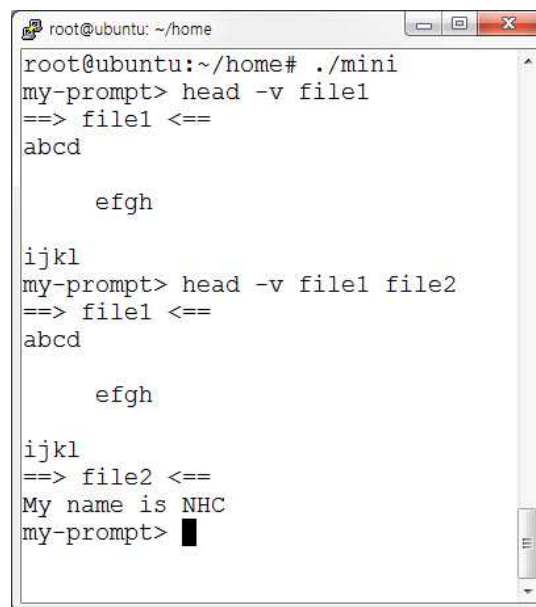
ijkl
root@ubuntu:~/home# head -v file1 file2
==> file1 <==
abcd

    efgh

ijkl

==> file2 <==
My name is NHC
```

[ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> head -v file1
==> file1 <==
abcd

    efgh

ijkl
my-prompt> head -v file1 file2
==> file1 <==
abcd

    efgh

ijkl
==> file2 <==
My name is NHC
my-prompt> █
```

-q

파일명을 함께 출력하지 않는다

[ linux ]

```
root@ubuntu:~/home# head -q file1 file2
abcd

    efgh

ijkl
My name is NHC
```

[결과]

```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> head -q file1 file2
abcd

efgh

ijkl
My name is NHC
my-prompt>

```

-n

출력을 원하는 줄 수를 정한다.

[ linux ]

```

root@ubuntu:~/home# head -n 1 file1
abcd
root@ubuntu:~/home# tail -n 1 file1
ijkl

```

[ 결과 ]

```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> head -n 1 file1
abcd
my-prompt> tail -n 1 file1
ijkl
my-prompt>

```

사용함수

1. os.path.exists(path)
  - 모듈이름 : "os"
  - 기능 : 파일/디렉토리가 존재하는가를 알아본다.
  - 인수 : path → 찾고자하는 파일/디렉토리
  - 반환값 : [성공] True  
[실패] False

2. open(file, mode)
  - 기능 : 파일작업을 하기위해 파일을 연다/
  - 인수 : file → 파일 이름  
mode → 파일을 열 때 사용하는 모드

모드	설명
r	읽기모드(디폴트)
r+	읽기+쓰기모드
w	쓰기 모드
a	쓰기+이어쓰기모드
t	텍스트모드(디폴트)
b	바이너리 모드

- 반환값 : [성공] 파일객체  
[실패] False

<구현설명>

chmod 명령어는 파일이나 디렉토리의 퍼미션 모드를 chmod()함수로 변경해주는 함수다. 숫자로 퍼미션 모드를 입력받을 수 도 있고, 문자로도 입력이 가능하다. 문자로 입력받는 경우 입력받은 문자를 캐릭터 단위로 검사하여 user, group, other에 해당하는 비트를 수정해주는 함수구현 사용하여 숫자 퍼미션으로 리턴해주는 값을 퍼미션 변경에 사용한다.

[ chmod [옵션][모드][파일명] ]

파일의 접근권한을 변경한다

[ linux ]

chmod

<테스트결과>

```

root@ubuntu: ~/home
root@ubuntu:~/home# ls -al
total 80
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx----- 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rw-r--r--  1 root root    4 Nov  4 06:08 file1
-rw-r--r--  1 root root   15 Nov  4 00:19 file2
-rw-r--r--  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rw-  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rw-  1 root root 15105 Nov  4 01:08 miniShell.c~
root@ubuntu:~/home# chmod 0700 file1
root@ubuntu:~/home# ls -al
total 80
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx----- 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rwx-----  1 root root    4 Nov  4 06:08 file1
-rw-r--r--  1 root root   15 Nov  4 00:19 file2
-rw-r--r--  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rw-  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rw-  1 root root 15105 Nov  4 01:08 miniShell.c~
root@ubuntu:~/home# █

```

```

root@ubuntu: ~/home
root@ubuntu:~/home# ls -al
total 80
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx----- 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rw-r--r--  1 root root    4 Nov  4 06:08 file1
-rw-r--r--  1 root root   15 Nov  4 00:19 file2
-rw-r--r--  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rw-  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rw-  1 root root 15105 Nov  4 01:08 miniShell.c~
root@ubuntu:~/home# chmod u=r,g-r,o+x file1
root@ubuntu:~/home# ls -al
total 80
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx----- 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-r-----r-x  1 root root    4 Nov  4 06:08 file1
-rw-r--r--  1 root root   15 Nov  4 00:19 file2
-rw-r--r--  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rw-  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rw-  1 root root 15105 Nov  4 01:08 miniShell.c~
root@ubuntu:~/home# █

```

## [ 결과 ]

```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls -al

drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx-----x 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rw-r--r-x  1 root root      4 Nov  4 06:08 file1
-rw-r--r-x  1 root root    15 Nov  4 00:19 file2
-rw-r--r-x  1 root root    27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> chmod 0700 file1
my-prompt> ls -al

drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx-----x 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rwx-----x 1 root root      4 Nov  4 06:08 file1
-rw-r--r-x  1 root root    15 Nov  4 00:19 file2
-rw-r--r-x  1 root root    27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> █
```

```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls -al

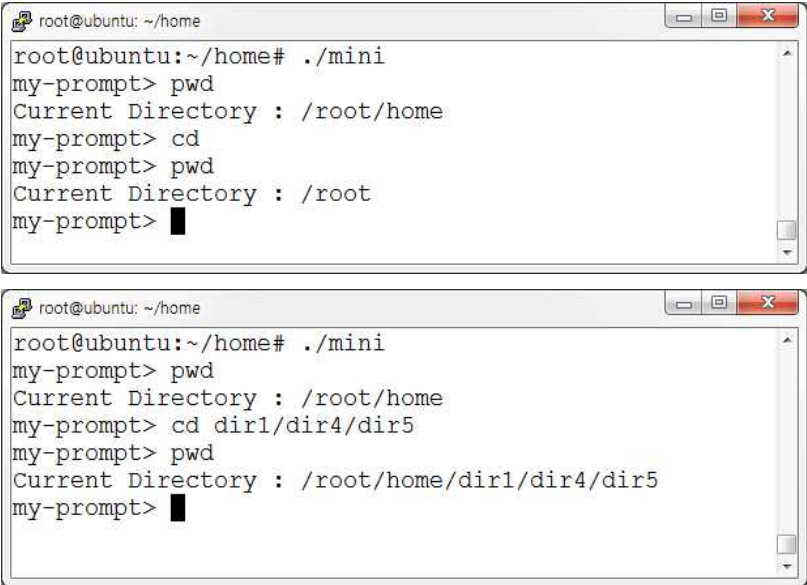
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx-----x 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-rw-r--r-x  1 root root      4 Nov  4 06:08 file1
-rw-r--r-x  1 root root    15 Nov  4 00:19 file2
-rw-r--r-x  1 root root    27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> chmod u=r,g-r,o+x file1
my-prompt> ls -al

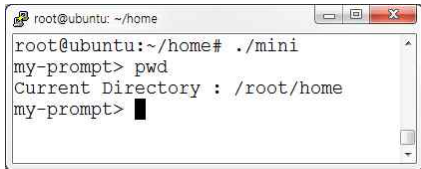
drwxr-xr-x  5 root root  4096 Nov  4 07:11 .
drwx-----x 19 root root  4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root  4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
-r-----r-x 1 root root      4 Nov  4 06:08 file1
-rw-r--r-x  1 root root    15 Nov  4 00:19 file2
-rw-r--r-x  1 root root    27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> █
```

	사용자기호	u(user),	소유자
		g(group),	그룹
		o(other),	다른사용자
		a(all)	소유자, 그룹, 사용자 모두 [기본]
	퍼미션기호	r(read)	읽기 권한을 준다
		w(write)	쓰기 권한을 준다
		x(excute)	실행 권한을 준다
		s (set user(group) ID)	파일 실행시 파일의 소유자 혹은 그룹 권한으로 실행한다.
		t (sticky bit)	sticky 비트를 설정한다
		u (user)	현재 소유자의 퍼미션 설정과 같은 내용으로 변경한다
		g (group)	현재 그룹의 퍼미션 설정과 같은 내용으로 변경한다
		o (other)	현재 다른 사용자의 퍼미션 설정과 같은 내용으로 변경한다
		l (locking)	강제로 파일을 잠근다
	설정기호	+	퍼미션 허가
		-	퍼미션 금지
		=	퍼미션 지정
	8진수	4000	파일 실행 시 파일 소유자 권한으로 실행된다
		2000	파일 실행 시 파일 그룹 권한으로 실행된다
		1000	sticky비트
		0400	소유자에게 읽기 권한을 준다
		0200	소유자에게 쓰기 권한을 준다
		0100	소유자에게 실행 권한을 준다
		0040	그룹에게 읽기 권한을 준다
		0020	그룹에게 쓰기 권한을 준다
		0010	그룹에게 실행 권한을 준다
		0004	다른 사용자에게 읽기 권한을 준다
		0002	다른 사용자에게 쓰기 권한을 준다
		0001	다른 사용자에게 실행 권한을 준다
사용함수	1. int chmod (const char *file, mode_t mode) ; - 헤더 : "sys/stat.h" - 기능 : 파일의 접근권한을 변경한다. - 인수 : char *file → 변경하려는 파일 이름 mode_t mode → 접근 권한 - 반환값 : [성공] 0 [실패] -1		

	<p>&lt;구현설명&gt;</p> <p>cd 명령어는 현재의 디렉토리에서 입력받은 디렉토리로 변경하거나 입력받지 않는다면 환경변수 'HOME' 에 해당하는 홈디렉토리로 chdir() 함수를 통해서 current directory를 변경한다.</p> <p>이 명령어는 Shell 내부에서 실행되어야 한다. 만약 fork된 child에서 실행된다면 자식프로세서의 currnet directory만 변경되고 부모프로세스인 Shell은 변경되지 않는다. 그래서 Shell code에 CD()함수를 추가했다.</p>
cd	<div> <div>[ cd [디렉토리 경로] ]</div> <div>디렉토리를 이동한다</div> </div> <p>[ linux ]</p> <pre> root@ubuntu:~/home# pwd /root/home root@ubuntu:~/home# cd dir1/dir4/dir5/ root@ubuntu:~/home/dir1/dir4/dir5# pwd /root/home/dir1/dir4/dir5 </pre> <p>[ 결과 ]</p>  <p>&lt;테스트결과&gt;</p>
사용함수	<p>1. int chdir(const char *path);</p> <ul style="list-style-type: none"> <li>- 헤더 : "unistd.h"</li> <li>- 기능 : Current Directory를 변경한다.</li> <li>- 인수 : const char *path → 변경을 하려는 폴더이름</li> <li>- 반환값 : [성공] 0 [실패] -1</li> </ul>



pwd	<구현설명>	pwd 명령어는 현재작업중인 위치의 디렉토리 이름을 알려주는 명령어다. Currnet Diretory값을 가져오는 getcwd()함수로 위치를 받아와 출력한다.	
	<테스트결과>	pwd	작업 디렉터리명을 출력한다
		<div>[ linux ]</div> <pre>root@ubuntu:~/home# pwd /root/home</pre> <div>[ 결과 ]</div> 	
사용함수	1. “unistd.h” char *getcwd(char *buf, size_tsize); - 기능 : 작업 디렉토리의 전체 이름을 구합니다. - 인수 : char *buf → [작업 디렉토리 문자열을 담을 버퍼] size_t size → 버퍼의 크기 - 반환값 : [성공]현재 작업 디렉토리 [실패] -1		



	<div>&lt;구현설명&gt;</div> <div><p>cp명령어는 각각의 파일들을 모두 open()하여 read(), write()함수의 반복으로 파일을 복사를 진행한다. 복사의 종류로는 3가지 복사가 가능하다.</p><ul style="list-style-type: none"><li>- file to file 은 단순히 복사될 파일 한 개와 한 개의 복사 할 파일이 존재하는 것이다.</li><li>- files to directory 은 여러개의 파일들을 입력한 후 마지막 인자로 디렉토리를 입력하면 모든 파일들이 디렉토리 내부에 복사되는 것이다.</li><li>- directory to directory는 디렉토리과 디렉토리간 복사로 디렉토리 내부의 모든 파일들이 복사되는 것이다. 이 복사는 'R' 옵션이 필요하다.</li></ul><p>복사를 하기 위해서는 복사하고자 하는 위치에 같은 이름이 있어서는 안되며 같은 이름이 존재 할 시 복사를 그냥 진행 할 지는 사용자에게 기본적으로 물어보도록 되어있다. 만약 물어보기 없이 진행하고 할 때는 'f' 옵션으로 진행하면 된다.</p></div>
cp	<div><div><div>[ cp [옵션] 원본파일 복사파일 ]</div><div>원본파일내용이 복사파일에 저장된다</div></div><div><div>[ linux ]</div><div><pre>root@ubuntu:~/home# ls dir1  dir2  dir3  file1  file2  file3 root@ubuntu:~/home# cat file1 abcd  efgh  ijkl root@ubuntu:~/home# cp file1 file4 root@ubuntu:~/home# cat file4 abcd  efgh  ijkl</pre></div><div><div>[ 결과 ]</div><div><div><div>root@ubuntu: ~/home</div><div>root@ubuntu:~/home# ./mini</div><div>my-prompt&gt; cat file1</div><div>abcd</div><div>efgh</div><div>ijkl</div><div>my-prompt&gt; cp file1 file4</div><div>my-prompt&gt; cat file4</div><div>abcd</div><div>efgh</div><div>ijkl</div><div>my-prompt&gt; </div></div><div><div>root@ubuntu: ~/home</div><div>root@ubuntu:~/home# ./mini</div><div>my-prompt&gt; ls dir3</div><div>file1  file2</div><div>my-prompt&gt; </div></div></div></div></div></div>

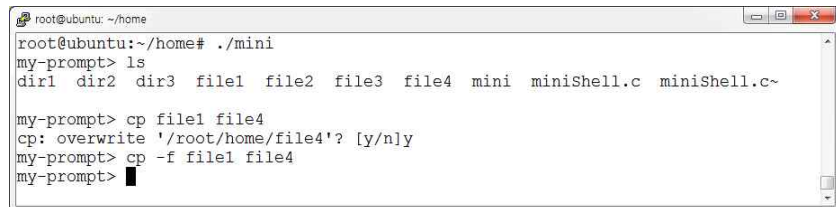
-f

복사 대상파일이 있으면 강제로 지우고 복사한다 [기본적으로 물어본다]

[ linux ]

```
root@ubuntu:~/home# ls
dir1 dir2 dir3 file1 file2 file3 file4
root@ubuntu:~/home# cp -f file1 file4
root@ubuntu:~/home# ls
dir1 dir2 dir3 file1 file2 file3 file4
```

[ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls
dir1 dir2 dir3 file1 file2 file3 file4 mini miniShell.c miniShell.c~

my-prompt> cp file1 file4
cp: overwrite '/root/home/file4'? [y/n]y
my-prompt> cp -f file1 file4
my-prompt> █
```

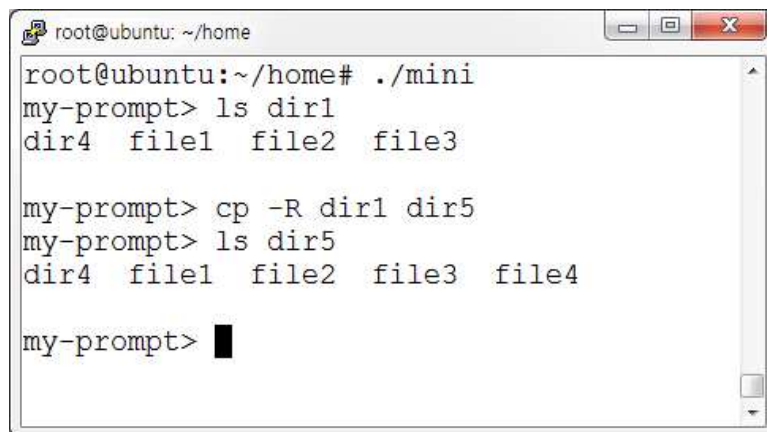
-R

디렉터리 복사 시 하위 디렉터리와 파일을 모두 복사 한다

[ linux ]

```
root@ubuntu:~/home# ls dir1
dir4 file1 file2 file3
root@ubuntu:~/home# cp -R dir1 dir5
root@ubuntu:~/home# ls dir5
dir4 file1 file2 file3
```

[ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls dir1
dir4 file1 file2 file3

my-prompt> cp -R dir1 dir5
my-prompt> ls dir5
dir4 file1 file2 file3 file4

my-prompt> █
```

사용함수

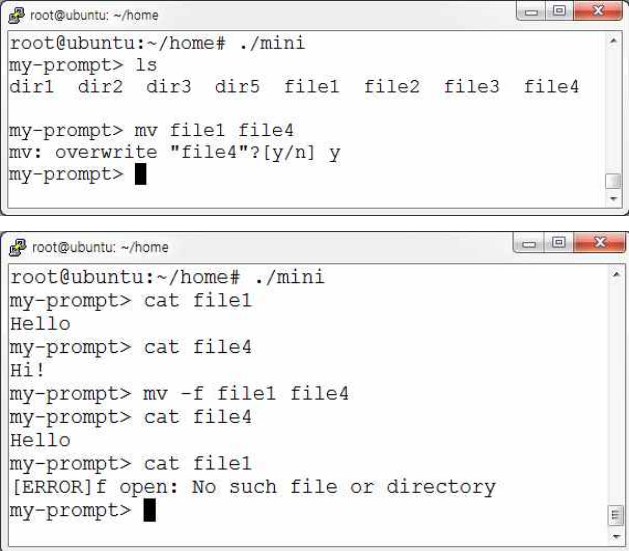
1. int stat(const char \*pathname, struct \_stat \*buf);
  - 헤더 : "sys/stat.h"
  - 기능 : 파일 정보를 확인하는 함수, 파일의 유무도 확인이 가능하다.
  - 인수 : const char \*pathname → 파일 또는 폴더 경로  
 struct \_stat \*buf → 구조체 stat 의 포인터
  - 반환값 : [성공] 0, buf인자에 stat구조체에 파일  
 [실패] -1
2. int open (const char \*FILENAME, int FLAGS[, mode\_t MODE])
  - 헤더 : "fcntl.h"
  - 기능 : 리눅스에서 제공하는 함수로 파일을 사용하기 위해 여러 모드로 열기한다.
  - 인수 : char \*FILENAME → 대상 파일 이름  
 int FLAGS → 파일에 대한 열기 옵션  
 [, mode\_t MODE] → O\_CREAT옵션 사용에 의해 파일이 생성될 때 지정되는 파일 접근 권한
  - 반환값 : [성공] 일 디스크립터의 양의 정수 값  
 [실패] -1

옵션	설명
RDONLY	읽기 전용으로 열기
O_WRONLY	쓰기 전용으로 열기
O_RDWR	읽기와 쓰기가 모두 가능
O_CREAT	해당 파일이 없으면 생성합니다. O_CREAT로 파일을 생성하게 된다면 파일의 접근권한을 지정하기 위해 접근 권한 값을 추가해야 합니다.
O_EXCL	O_CREAT를 사용할 때, O_EXCL를 함께 사용하면, 이미 파일이 있을 때에는 open() 되지 않아 이전 파일을 보존할 수 있습니다.
O_TRUNC	기존의 파일 내용을 모두 삭제합니다.
O_APPEND	파일을 추가하여 쓰기가 되도록 open 후에 쓰기 포인터가 파일의 끝에 위치하게 됩니다.
O_NOCITTY	열기 대상이 터미널일 경우, 이 터미널이 프로그램의 제어 터미널로 할당하지 않습니다.
O_NONBLOCK	읽을 내용이 없을 때에는 읽을 내용이 있을 때까지 기다리지 않고 바로 복귀합니다.
O_SYNC	쓰기를 할 때, 실제 쓰기가 완료될 때 까지 기다립니다. 즉, 물리적으로 쓰기가 완료되어야 복귀하게 됩니다.

3. ssize\_t read (int fd, void \*buf, size\_t nbytes)
  - 헤더 : "unistd.h"
  - 기능 : open() 함수로 열기를 한 파일의 내용을 읽기를 한다.
  - 인수 : int fd → 파일 디스크립터  
 void \*buf → 파일을 읽어 들일 버퍼  
 size\_t nbytes → 버퍼의 크기
  - 반환값 : [성공] 읽어들이는 바이트 수  
 [실패] -1
4. ssize\_t write (int fd, const void \*buf, size\_t n)
  - 헤더 : "unistd.h"
  - 기능 : open() 함수로 열기를 한 파일에 내용을 쓰기 한다.
  - 인수 : int fd → 파일 디스크립터  
 void \*buf → 파일에 쓰기를 할 내용을 담은 버퍼  
 size\_t n → 쓰기할 바이트 수
  - 반환값 : [성공] 쓰기 한 바이트 수  
 [실패] -1

5. int close (int fd)
  - 헤더 : "unistd.h"
  - 기능 : open() 함수로 열기를 한 파일의 사용을 중지한다.
  - 인수 : int fd → 파일 디스크립터
  - 반환값 : [성공] 0  
[실패] -1
6. DIR \*opendir(const char \*name);
  - 헤더 : "dirent.h"
  - 기능 : 지정한 디렉토리 열기를 합니다. 디렉토리 안의 내용을 보기 위함 이다.
  - 인수 : char \*name → 열기 대상 디렉토리
  - 반환값 : [성공] 디렉토리 정보 구조체인 DIR 포인터  
[실패] NULL
7. struct dirent \*readdir( DIR \*dir);
  - 헤더 : "dirent.h"
  - 기능 : 열기를 한 디렉토리 안에 있는 파일 중 첫번째 파일에 대한 정보를 구합니다.
  - 인수 : DIR \*dir → opendir()에서 열기한 디렉토리 정보
  - 반환값 : [성공] 파일이나 디렉토리 정보  
[실패] NULL
8. int stat(const char \*pathname, struct \_stat \*buf);
  - 헤더 : "sys/stat.h"
  - 기능 : 파일 정보를 확인하는 함수, 파일의 유무도 확인이 가능하다.
  - 인수 : const char \*pathname → 파일 또는 폴더 경로  
struct \_stat \*buf → 구조체 stat 의 포인터
  - 반환값 : [성공] 0, buf인자에 stat구조체에 파일  
[실패] -1

	<div>&lt;구현설명&gt;</div>	<div>mv명령어는 파일 혹은 파일들을 해당 디렉토리에 옮기거나 파일이름으로 옮긴다. 이 때 옮긴다는 의미는 파일을 open()하여 read()하고 옮기려고 하는 위치에 write()함으로서 복사생성 한 후 복사한 파일을 unlink로 지움으로서 옮긴 것처럼 보이게 한다. 이동 시 이미 존재하는 파일이름이라면 기본적으로 물어본다. 하지만 'f'옵션은 이 물어보기를 금지하고 강제로 이동시킨다.</div>
mv	<div>&lt;테스트결과&gt;</div>	<div><div>[ mv [옵션] 원본파일 이동경로 ]</div><div>파일을 옮기거나 이름을 바꾼다</div><div><div>[ linux ]</div><div>root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file4 root@ubuntu:~/home# mv file4 dir3 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 root@ubuntu:~/home# ls dir3 file4</div><div>-----</div><div>[change name]</div><div>root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file4 root@ubuntu:~/home# mv file4 file5 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file5</div><div><div>[ 결과 ]</div><div><div>root@ubuntu: ~/home</div><div>root@ubuntu:~/home# ./mini my-prompt&gt; ls dir1 dir2 dir3 dir5 file1 file2 file3 file4 mini miniShell.c miniShell.c~ my-prompt&gt; mv file4 file5 my-prompt&gt; ls dir1 dir2 dir3 dir5 file1 file2 file3 file5 mini miniShell.c miniShell.c~ my-prompt&gt; </div><div><div>root@ubuntu: ~/home</div><div>root@ubuntu:~/home# ./mini my-prompt&gt; ls dir1 dir2 dir3 dir5 file1 file2 file3 file4 mini miniShell.c miniShell.c~ my-prompt&gt; mv file4 dir3 my-prompt&gt; ls dir3 file4 my-prompt&gt; </div></div></div></div></div></div>

		-f	덮어쓸 때 묻지 않는다 [기본적으로 물어본다]
		<pre> [ linux ]  root@ubuntu:~/home# cat file1 Hello root@ubuntu:~/home# cat file4 Hi! root@ubuntu:~/home# mv -f file1 file4 root@ubuntu:~/home# cat file4 Hello root@ubuntu:~/home# cat file1 cat: file1: No such file or directory </pre> <p>[ 결과 ]</p> 	
사용 함수	<ol style="list-style-type: none"> <li>1. int stat(const char *pathname, struct _stat *buf); <ul style="list-style-type: none"> <li>- 헤더 : "sys/stat.h"</li> <li>- 기능 : 파일 정보를 확인하는 함수, 파일의 유무도 확인이 가능하다.</li> <li>- 인수 : const char *pathname → 파일 또는 폴더 경로 struct _stat *buf → 구조체 stat 의 포인터</li> <li>- 반환값 : [성공] 0, buf인자에 stat구조체에 파일 [실패] -1</li> </ul> </li> <li>2. DIR *opendir(const char *name); <ul style="list-style-type: none"> <li>- 헤더 : "dirent.h"</li> <li>- 기능 : 지정한 디렉토리 열기를 합니다. 디렉토리 안의 내용을 보기 위함 이다.</li> <li>- 인수 : char *name → 열기 대상 디렉토리</li> <li>- 반환값 : [성공] 디렉토리 정보 구조체인 DIR 포인터 [실패] NULL</li> </ul> </li> <li>3. struct dirent *readdir( DIR *dir); <ul style="list-style-type: none"> <li>- 헤더 : "dirent.h"</li> <li>- 기능 : 열기를 한 디렉토리 안에 있는 파일 중 첫번째 파일에 대한 정보를 구합니다.</li> <li>- 인수 : DIR *dir → opendir()에서 열기한 디렉토리 정보</li> <li>- 반환값 : [성공] 파일이나 디렉토리 정보 [실패] NULL</li> </ul> </li> <li>4. int unlink(const char *path) <ul style="list-style-type: none"> <li>- 헤더 : "unistd.h"</li> <li>- 기능 : path지정된 파일과 디렉토리의 inode에서 링크 수를 감소시킨다. 링크수가 0이되면 path에 지정된 파일이 삭제된다.</li> </ul> </li> </ol>		

- 인수 : const char \*path → 삭제할 파일의 경로
- 반환값 : [성공] 0  
[실패] -1

#### 5. int open (const char \*FILENAME, int FLAGS[, mode\_t MODE])

- 헤더 : "fcntl.h"
- 기능 : 리눅스에서 제공하는 함수로 파일을 사용하기 위해 여러 모드로 열기한다.
- 인수 : char \*FILENAME → 대상 파일 이름  
int FLAGS → 파일에 대한 열기 옵션  
[, mode\_t MODE] → O\_CREAT 옵션 사용에 의해 파일이 생성될 때 지정되는 파일 접근 권한
- 반환값 : [성공] 일 디스크립터의 양의 정수 값  
[실패] -1

옵션	설명
RDONLY	읽기 전용으로 열기
O_WRONLY	쓰기 전용으로 열기
O_RDWR	읽기와 쓰기가 모두 가능
O_CREAT	해당 파일이 없으면 생성합니다. O_CREAT로 파일을 생성하게 된다면 파일의 접근권한을 지정하기 위해 접근 권한 값을 추가해야 합니다.
O_EXCL	O_CREAT를 사용할 때, O_EXCL를 함께 사용하면, 이미 파일이 있을 때에는 open() 되지 않아 이전 파일을 보존할 수 있습니다.
O_TRUNC	기존의 파일 내용을 모두 삭제합니다.
O_APPEND	파일을 추가하여 쓰기가 되도록 open 후에 쓰기 포인터가 파일의 끝에 위치하게 됩니다.
O_NOCTTY	열기 대상이 터미널일 경우, 이 터미널이 프로그램의 제어 터미널로 할당하지 않습니다.
O_NONBLOCK	읽을 내용이 없을 때에는 읽을 내용이 있을 때까지 기다리지 않고 바로 복귀합니다.
O_SYNC	쓰기를 할 때, 실제 쓰기가 완료될 때 까지 기다립니다. 즉, 물리적으로 쓰기가 완료되어야 복귀하게 됩니다.

#### 6. ssize\_t read (int fd, void \*buf, size\_t nbytes)

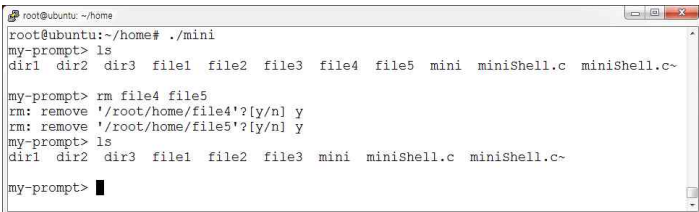
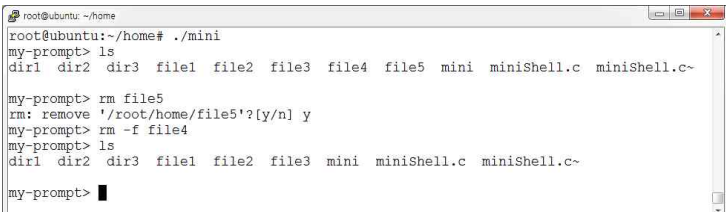

- 헤더 : "unistd.h"
- 기능 : open() 함수로 열기를 한 파일의 내용을 읽기를 한다.
- 인수 : int fd → 파일 디스크립터  
void \*buf → 파일을 읽어 들일 버퍼  
size\_t nbytes → 버퍼의 크기
- 반환값 : [성공] 읽어들이는 바이트 수  
[실패] -1

#### 7. ssize\_t write (int fd, const void \*buf, size\_t n)

- 헤더 : "unistd.h"
- 기능 : open() 함수로 열기를 한 파일에 내용을 쓰기 한다.
- 인수 : int fd → 파일 디스크립터  
void \*buf → 파일에 쓰기를 할 내용을 담은 버퍼  
size\_t n → 쓰기할 바이트 수
- 반환값 : [성공] 쓰기 한 바이트 수  
[실패] -1

#### 8. int close (int fd)

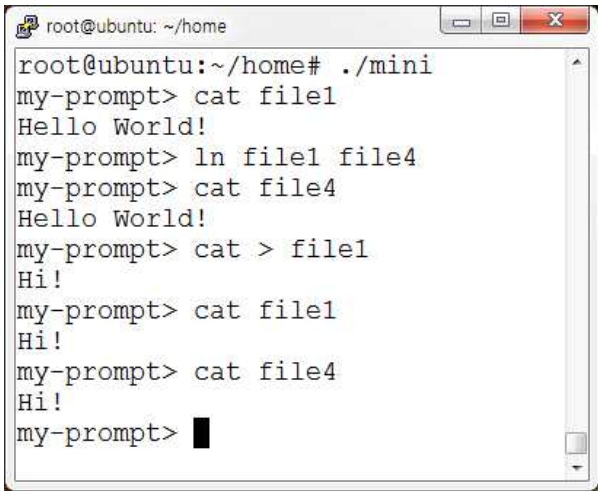
- 헤더 : "unistd.h"
- 기능 : open() 함수로 열기를 한 파일의 사용을 중지한다.
- 인수 : int fd → 파일 디스크립터
- 반환값 : [성공] 0  
[실패] -1

rm	<구현설명>	rm 명령어는 파일과 파일들을 지우거나 디렉토리를 전부 지운다. 이때 디렉토리를 지우고자 할 때는 디렉토리를 지우는 함수를 재귀적으로 수행하면서 내부 파일들을 모두 unlink()로 지운 후 디렉토리를 remove()로 지운다. 디렉토리를 지우고자 할 때는 'r' 옵션을 사용해야 한다.	
	<테스트결과>	[ rm [옵션] 파일.. ]	파일을 지운다
		<div>[ 결과 ]</div> 	
		-f	강제로 지운다
		<div>[ linux ]</div> <pre>root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 file4 root@ubuntu:~/home# rm -f file4 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3</pre> <div>[ 결과 ]</div> 	
	-r	하위 디렉토리를 포함하여 모두 지운다	
		<div>[ linux ]</div> <pre>root@ubuntu:~/home# mkdir -p dir4/dir5 root@ubuntu:~/home# rm dir4 rm: cannot remove 'dir4': Is a directory root@ubuntu:~/home# rm -r dir4 root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3</pre> <div>[ 결과 ]</div> 	



사용 함수

1. int stat(const char \*pathname, struct \_stat \*buf);
  - 헤더 : "sys/stat.h"
  - 기능 : 파일 정보를 확인하는 함수, 파일의 유무도 확인이 가능하다.
  - 인수 : const char \*pathname → 파일 또는 폴더 경로  
 struct \_stat \*buf → 구조체 stat 의 포인터
  - 반환값 : [성공] 0, buf인자에 stat구조체에 파일  
 [실패] -1
2. DIR \*opendir(const char \*name);
  - 헤더 : "dirent.h"
  - 기능 : 지정한 디렉토리 열기를 합니다. 디렉토리 안의 내용을 보기 위함 이다.
  - 인수 : char \*name → 열기 대상 디렉토리
  - 반환값 : [성공] 디렉토리 정보 구조체인 DIR 포인터  
 [실패] NULL
3. struct dirent \*readdir( DIR \*dir);
  - 헤더 : "dirent.h"
  - 기능 : 열기를 한 디렉토리 안에 있는 파일 중 첫번째 파일에 대한 정보를 구합니다.
  - 인수 : DIR \*dir → opendir()에서 열기한 디렉토리 정보
  - 반환값 : [성공] 파일이나 디렉토리 정보  
 [실패] NULL
4. int unlink( const char \*path )
  - 헤더 : "unistd.h"
  - 기능 : path지정된 파일과 디렉토리의 inode에서 링크 수를 감소시킨다.  
 링크수가 0이되면 path에 지정된 파일이 삭제된다.
  - 인수 : const char \*path → 삭제할 파일의 경로
  - 반환값 : [성공] 0  
 [실패] -1
5. char \*fgets(char \*str, int size, FILE \*stream);
  - 헤더 : "stdio.h"
  - 기능 : 파일 스트림으로부터 문자열을 읽는다.
  - 인수 : int \*str → 문자열을 읽어 들일 메모리 포인터  
 int size → 입력 받을 수 있는 최대 문자 개수  
 FILE \*stream → 읽기를 하고자 하는 FILE 포인터
  - 반환값 : [성공] 메모리 포인터  
 [실패] NULL
6. int remove(const char \*pathname);
  - 헤더 : "stdio.h "
  - 기능 : 파일 또는 디렉토리를 삭제한다.
  - 인수 : char \*pathname → 삭제할 파일이나 디렉토리
  - 반환값 : [성공] 0  
 [실패] -1

ln	<구현설명>	ln명령어는 inode자체를 링크하는 hard link와 파일의 path만을 link하는 soft link를 생성 할 수 있다. hard link는 link()함수로 가능하다. soft link는 symlink()함수를 사용하여 가능하다.	
		[ ln [옵션] 원본 [대상] ]	파일 링크 명령어 [하드 링크]
	<테스트결과>	<p>[ linux ]</p> <pre> root@ubuntu:~/home# cat file1 Hello World! root@ubuntu:~/home# ln file1 file4 root@ubuntu:~/home# cat file4 Hello World! root@ubuntu:~/home# cat &gt; file1 Hi! ^C root@ubuntu:~/home# cat file1 Hi! root@ubuntu:~/home# cat file4 Hi! </pre> <p>[ 결과 ]</p> 	

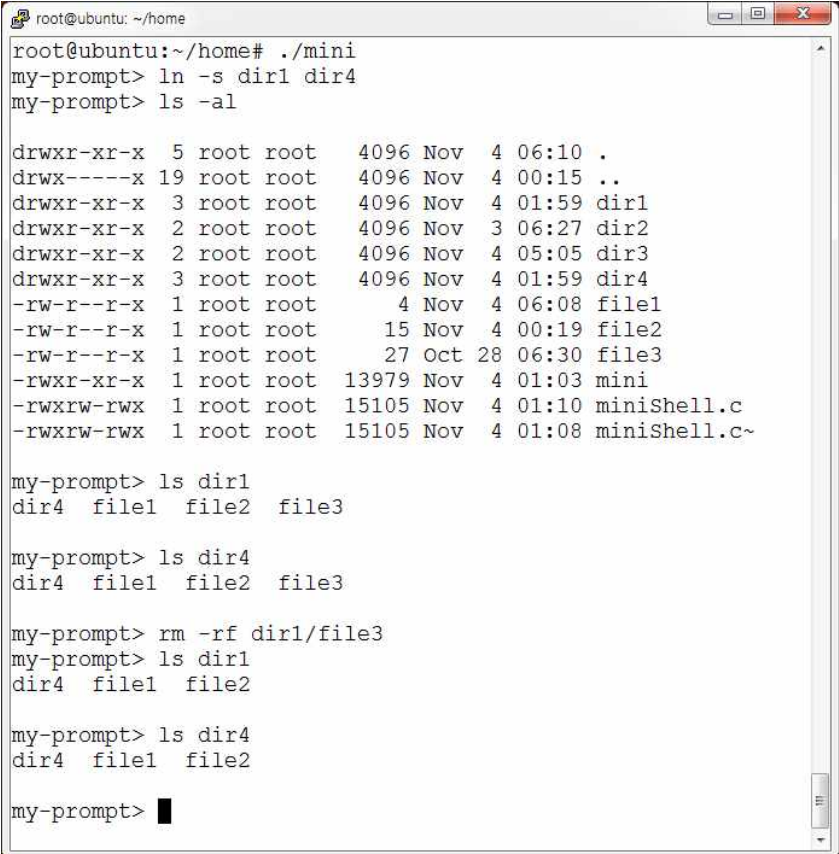
-s

링크할 원본이 심볼릭 링크된 파일이면 그 파일을 링크한다 [소프트 링크]

#### [ linux ]

```
root@ubuntu:~/home# ln -s dir1 dir4
root@ubuntu:~/home# ls
dir1 dir2 dir3 dir4 file1 file2 file3
root@ubuntu:~/home# ls -al
total 24
drwxr-xr-x  5 root root 4096 Oct  8 09:24 .
drwx----- 17 root root 4096 Oct  8 09:22 ..
drwxr-xr-x  3 root root 4096 Oct  8 08:40 dir1
drwxr-xr-x  2 root root 4096 Oct  8 09:00 dir2
drwxr-xr-x  2 root root 4096 Oct  8 07:21 dir3
lrwxrwxrwx  1 root root   4 Oct  8 09:24 dir4 -> dir1
-rw-r--r--  1 root root   4 Oct  8 09:23 file1
-rw-r--r--  1 root root   0 Oct  8 07:21 file2
-rw-r--r--  1 root root   0 Oct  8 07:22 file3
root@ubuntu:~/home# ls dir1
dir4 file1 file2 file3
root@ubuntu:~/home# ls dir4
dir4 file1 file2 file3
root@ubuntu:~/home# rm -rf dir1/file3
root@ubuntu:~/home# ls dir1
dir4 file1 file2
root@ubuntu:~/home# ls dir4
dir4 file1 file2
```

#### [ 결과 ]



```
root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ln -s dir1 dir4
my-prompt> ls -al

drwxr-xr-x  5 root root  4096 Nov  4 06:10 .
drwx-----x 19 root root  4096 Nov  4 00:15 ..
drwxr-xr-x  3 root root  4096 Nov  4 01:59 dir1
drwxr-xr-x  2 root root  4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root  4096 Nov  4 05:05 dir3
drwxr-xr-x  3 root root  4096 Nov  4 01:59 dir4
-rw-r--r-x  1 root root    4 Nov  4 06:08 file1
-rw-r--r-x  1 root root   15 Nov  4 00:19 file2
-rw-r--r-x  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrw-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrw-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> ls dir1
dir4 file1 file2 file3

my-prompt> ls dir4
dir4 file1 file2 file3

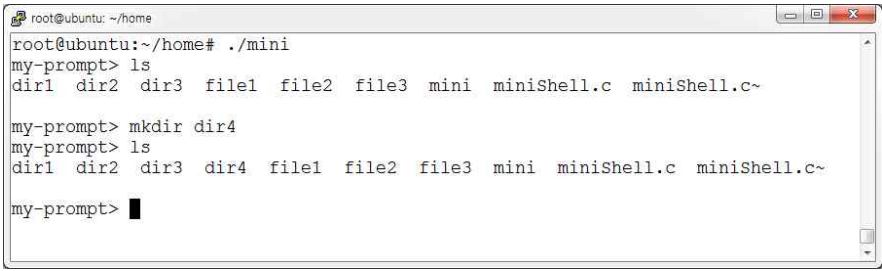

my-prompt> rm -rf dir1/file3
my-prompt> ls dir1
dir4 file1 file2

my-prompt> ls dir4
dir4 file1 file2

my-prompt>
```

사용함수

1. int symlink( const char \*oldpath, const char \*newpath)
  - 헤더 : "unistd.h"
  - 기능 : 심볼릭 링크를 생성한다. (소프트 링크)
  - 인수 : char \*oldpath → 이미 존재하는 파일 이름  
char \*newpath → 만들고자하는 링크 이름
  - 반환값 : [성공] 0  
[실패] -1
2. int link( const char \*oldpath, const char \*newpath)
  - 헤더 : "unistd.h"
  - 기능 : 다른 이름으로 접근이 가능한 링크를 생성한다. (하드 링크)
  - 인수 : char \*oldpath → 이미 존재하는 파일 이름  
char \*newpath → 만들고자하는 링크 이름
  - 반환값 : [성공] 0  
[실패] -1

	<구현설명>	<p>mkdir 명령어는 파이썬으로 만들었으며 디렉토리를 os.mkdir() 함수로 생성하는 명령어다.</p> <p>다수의 디렉터리 이름을 입력받아 생성이 가능하다. 'm' 옵션은 디렉터리 생성시 기본 퍼미션 모드인 '0755' 대신에 다른 퍼미션 모드를 주어 생성이 가능하도록 한다. 'p' 옵션은 입력한 디렉터리의 상위 디렉터리까지 모두 os.makedirs() 함수를 사용하여 생성할 수 있도록 하였다.</p>
mkdir	<테스트결과>	<div data-bbox="531 488 1222 521">[ mkdir [옵션] 디렉터리이름.. ]    디렉터리를 생성한다</div> <div data-bbox="531 521 643 555">[ linux ]</div> <div data-bbox="531 566 1074 678"> <pre> root@ubuntu:~/home# ls dir1 dir2 dir3 file1 file2 file3 root@ubuntu:~/home# mkdir dir4 root@ubuntu:~/home# ls dir1 dir2 dir3 dir4 file1 file2 file3 </pre> </div> <div data-bbox="531 734 632 768">[ 결과 ]</div> <div data-bbox="531 779 1417 1048">  <pre> root@ubuntu: ~/home root@ubuntu:~/home# ./mini my-prompt&gt; ls dir1 dir2 dir3 file1 file2 file3 mini miniShell.c miniShell.c~  my-prompt&gt; mkdir dir4 my-prompt&gt; ls dir1 dir2 dir3 dir4 file1 file2 file3 mini miniShell.c miniShell.c~  my-prompt&gt; █ </pre> </div> <div data-bbox="531 1059 1417 1305">  <pre> root@ubuntu: ~/home root@ubuntu:~/home# ./mini my-prompt&gt; ls dir1 dir2 dir3 file1 file2 file3 mini miniShell.c miniShell.c~  my-prompt&gt; mkdir dir4 dir5 my-prompt&gt; ls dir1 dir2 dir3 dir4 dir5 file1 file2 file3 mini miniShell.c miniShell.c~  my-prompt&gt; █ </pre> </div>
		<div data-bbox="531 1541 1353 1597">-m                                    새로 만드는 디렉터리의 권한을 설정 한다</div> <div data-bbox="531 1641 643 1675">[ linux ]</div> <div data-bbox="531 1686 1117 1977"> <pre> root@ubuntu:~/home# ls -al total 24 drwxr-xr-x  5 root root 4096 Oct  8 09:32 . drwx----- 17 root root 4096 Oct  8 09:22 .. drwxr-xr-x  3 root root 4096 Oct  8 09:25 dir1 drwxr-xr-x  2 root root 4096 Oct  8 09:00 dir2 drwxr-xr-x  2 root root 4096 Oct  8 07:21 dir3 -rw-r--r--  1 root root   4 Oct  8 09:23 file1 -rw-r--r--  1 root root   0 Oct  8 07:21 file2 -rw-r--r--  1 root root   0 Oct  8 07:22 file3 root@ubuntu:~/home# mkdir -m 0700 dir4 root@ubuntu:~/home# ls -al   grep dir4 drwx-----  2 root root 4096 Oct  8 09:33 dir4 </pre> </div>

## [ 결과 ]

```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls
dir1 dir2 dir3 file1 file2 file3 mini miniShell.c miniShell.c~

my-prompt> mkdir -m 0700 dir4
my-prompt> ls -al
drwxr-xr-x  6 root root 4096 Nov  4 06:23 .
drwx-----x 19 root root 4096 Nov  4 06:21 ..
drwxr-xr-x  3 root root 4096 Nov  4 06:11 dir1
drwxr-xr-x  2 root root 4096 Nov  3 06:27 dir2
drwxr-xr-x  2 root root 4096 Nov  4 05:05 dir3
drwx-----x  2 root root 4096 Nov  4 06:23 dir4
-rw-r--r-x  1 root root    4 Nov  4 06:08 file1
-rw-r--r-x  1 root root   15 Nov  4 00:19 file2
-rw-r--r-x  1 root root   27 Oct 28 06:30 file3
-rwxr-xr-x  1 root root 13979 Nov  4 01:03 mini
-rwxrwx-rwx  1 root root 15105 Nov  4 01:10 miniShell.c
-rwxrwx-rwx  1 root root 15105 Nov  4 01:08 miniShell.c~

my-prompt> █

```

-p

상위 경로도 함께 생성한다

## [ linux ]

```

root@ubuntu:~/home# mkdir -p dir4/dir5/
root@ubuntu:~/home# ls dir4
dir5

```

## [ 결과 ]

```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> ls
dir1 dir2 dir3 file1 file2 file3 mini miniShell.c miniShell.c~

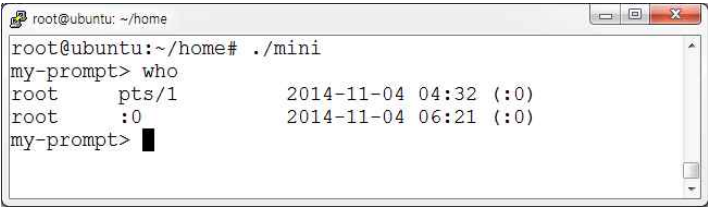
my-prompt> mkdir -p dir4/dir5/
my-prompt> ls dir4
dir5

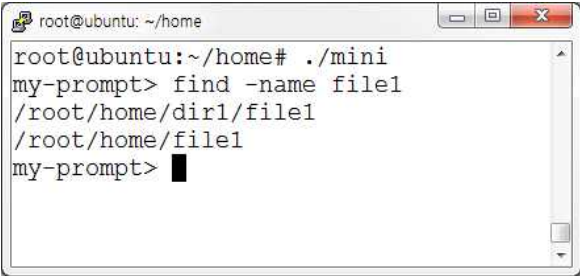
my-prompt> █

```

## 사용 함수

1. `os.path.isdir(path)`
  - 모듈이름 : "os"
  - 기능 : 경로가 디렉터리인지 아닌지 검사한다.
  - 인수 : `path` → 확인하려는 디렉토리
  - 반환값 : [성공] `True`  
[실패] `False`
2. `os.makedirs(path[, mode])`
  - 모듈이름 : "os"
  - 기능 : 인자로 전달된 디렉터리경로 전부를 생성한다.  
즉, 부모 디렉터리도 함께 생성한다
  - 인수 : `path` → 생성하려는 디렉토리  
: `mode` → Permission Mode
  - 반환값 : [성공] `True`  
[실패] `False`
3. `os.mkdir(path[, mode])`
  - 모듈이름 : "os"
  - 기능 : 디렉터리를 생성한다
  - 인수 : `path` → 생성하려는 디렉토리  
: `mode` → Permission Mode
  - 반환값 : [성공] `True`  
[실패] `False`

who	<구현설명>	who 명령어는 현재 접속되어 있는 user들의 계정명, 터미널 정보, 접속시간을 utmp파일을 getutxent()함수로 하나하나 읽어 참조하여 알 수 있다.	
	<테스트결과>	<div>[ who ]</div> <div>호스트에 로그인한 사용자 정보를 출력한다</div> <div>[ linux ]</div> <pre> root@ubuntu:~/home# who root      :0                2014-10-02 21:57 (:0) root      pts/13           2014-10-02 21:57 (:0) root      pts/15           2014-10-08 07:23 (192.168.200.102) root      pts/16           2014-10-08 07:23 (192.168.200.102) root      pts/18           2014-10-08 08:32 (192.168.200.100) root      pts/26           2014-10-08 08:40 (192.168.200.100) </pre> <div>[ 결과 ]</div> 	
사용함수	1. struct utmpx *getutxent(void); <ul style="list-style-type: none"> <li>- 헤더 : "utmpx.h"</li> <li>- 기능 : /var/adm/utmpx 파일에서 로그인 정보를 순차적으로 읽어온다</li> <li>- 반환값 : [성공] utmpx 구조체 (로그인 정보) [실패] NULL</li> </ul>		

find	<구현설명>	<p>find 명령어는 찾고자 하는 파일과 디렉토리를 currnet working directory로 부터 이하 디렉토리를 방문한다. 그러면서 엔트리를 검색하여 찾고자 하는 파일 및 디렉토리를 찾아준다.</p> <p>총 3가지 옵션을 사용 가능하다.</p> <ul style="list-style-type: none"> <li>- 'name' 옵션사용시 해당하는 이름의 파일및 디렉토리를 찾아준다.</li> <li>- 'type' 옵션사용시 해당하는 타입의 파일및 디렉토리를 찾아준다.</li> <li>- 'user' 옵션사용시 해당하는 user의 파일및 디렉토리를 찾아준다.</li> </ul>	
	<테스트결과>	[ find [패스] [옵션] [작업] ]	주어진 조건을 검색하여 파일을 찾는다
		-name	지정된 이름의 파일을 찾는다
		<p>[ linux ]</p> <pre>root@ubuntu:~/home# find -name file1 ./dir1/file1 ./file1</pre> <p>[ 결과 ]</p> 	
		-user	user 소유의 파일을 찾는다
		<p>[ linux ]</p> <pre>root@ubuntu:~/home# find -user root . ./file3 ./file2 ./dir3 ./dir3/file4 ./dir1 ./dir1/file2 ./dir1/dir4 ./dir1/dir4/dir5 ./dir1/file1 ./dir2 ./dir2/dir1 ./file1</pre> <p>[ 결과 ]</p>	



```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> find -user root
/root/home/file3
/root/home/miniShell.c
/root/home/file2
/root/home/miniShell.c~
/root/home/mini
/root/home/dir3/file4
/root/home/dir3
/root/home/dir1/file2
/root/home/dir1/dir4/dir5
/root/home/dir1/dir4
/root/home/dir1/file1
/root/home/dir1
/root/home/dir2/dir1
/root/home/dir2
/root/home/file1
my-prompt> █

```

-type

지정된 형식의 파일을 찾는다  
b(블록파일), c(문자), d(디렉토리)  
, f(파일), l(링크파일), s(소켓)

#### [ linux ]

```

root@ubuntu:~/home# find -type d
.
./dir3
./dir1
./dir1/dir4
./dir1/dir4/dir5
./dir2
./dir2/dir1
root@ubuntu:~/home# █

```

#### [ 결과 ]

```

root@ubuntu: ~/home
root@ubuntu:~/home# ./mini
my-prompt> find -type d
/root/home/dir3
/root/home/dir1/dir4/dir5
/root/home/dir1/dir4
/root/home/dir1
/root/home/dir2/dir1
/root/home/dir2
my-prompt> █

```

사용함수

1. char \*getcwd(char \*buf, size\_t size);
  - 헤더 : "unistd.h"
  - 기능 : 작업 디렉토리의 전체 이름을 구합니다.
  - 인수 : char \*buf → 작업 디렉토리 문자열을 담을 버퍼  
size\_t size → 버퍼의 크기
  - 반환값 : [성공] 현재 작업 디렉토리  
[실패] -1
2. DIR \*opendir(const char \*name);
  - 헤더 : "dirent.h"
  - 기능 : 지정한 디렉토리 열기를 합니다. 디렉토리 안의 내용을 보기 위함 이다.
  - 인수 : char \*name → 열기 대상 디렉토리
  - 반환값 : [성공] 디렉토리 정보 구조체인 DIR 포인터  
[실패] NULL
3. struct dirent \*readdir( DIR \*dir);
  - 헤더 : "dirent.h"
  - 기능 : 열기를 한 디렉토리 안에 있는 파일 중 첫번째 파일에 대한 정보를 구합니다.
  - 인수 : DIR \*dir → opendir()에서 열기한 디렉토리 정보
  - 반환값 : [성공] 파일이나 디렉토리 정보  
[실패] NULL
4. int stat(const char \*pathname, struct \_stat \*buf);
  - 헤더 : "sys/stat.h"
  - 기능 : 파일 정보를 확인하는 함수, 파일의 유무도 확인이 가능하다.
  - 인수 : const char \*pathname → 파일 또는 폴더 경로  
struct \_stat \*buf → 구조체 stat 의 포인터
  - 반환값 : [성공] 0, buf인자에 stat구조체에 파일  
[실패] -1

## \* 별첨 [ Source Code ]

### minishlib.h

```
/*-----  
고급시스템 프로그래밍 : minishlib.h  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*      커맨드 사용에 필요한 헤더들과 define정의 코드  
*/  
  
#include <sys/stat.h>  
#include <sys/types.h>  
#include <unistd.h>  
#include <time.h>  
#include <dirent.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <pwd.h>  
#include <grp.h>  
#include <utmpx.h>  
#include <fcntl.h>  
#include <ftw.h>  
#include <errno.h>  
#include <utime.h>  
  
#define TRUE 1  
#define FALSE 0  
  
#define NO "n\n"  
#define YES "y\n"  
  
#define A 'a'  
#define B 'b'  
#define N 'n'  
#define smallR 'r'  
#define bigR 'R'  
#define F 'f'  
#define H 'h'  
#define I 'i'
```

```
#define S 's'  
#define L 'l'  
  
#define MAX_SIZE 100  
#define CHAR_PERM_SIZE 11  
#define TIME_SIZE 20
```

## ls.c

```
/*-----
고급시스템 프로그래밍 : Command [ ls ]

                                국민대학교 컴퓨터공학과
                                20093284 나홍철
-----*/

/*
 *      ls명령어를 사용하면 현재의 path를 기준으로 존재하는 엔트리들의
 *      정보를 화면에 출력할 수 있다.
 */

#include "minishlib.h"

/*      lFlag -> 긴리스트의 포맷으로 출력, aFlag -> 경로안의 모든 내용을 출력한다,
      rFlag -> 정렬의 순서를 역방향으로 한다, RFlag -> 현재 디렉터리를 기준으로 모든 하위의 디렉
      터리를 출력 */
int lFlag=FALSE, aFlag=FALSE, rFlag=FALSE, RFlag=FALSE;

/* Time output standard */
char *outpT = "%b %e %H:%M";

/* 숫자 퍼미션 모드를 인자로 받으면 인자로 받은 스트링변수에 반환하는 함수다 */
int StrMode(char pMode[], mode_t stmode)
{
    int mode;
    mode = stmode & S_IFMT; // Kind of Mode
    switch (mode)
    {
        // Directory
        case S_IFDIR:
            pMode[0] = 'd';
            break;

        // Block Device
        case S_IFBLK:
            pMode[0] = 'b';
            break;

        // Char Device
        case S_IFCHR:
            pMode[0] = 'c';
            break;
    }
}
```

```

        // Symbol
    case S_IFLNK:
        pMode[0] = 'l';
        break;
    }
    if (stmode & S_ISUID)
        pMode[3] = 's';
    if (stmode & S_ISGID)
        pMode[6] = 's';
    if (stmode & S_ISVTX)
        pMode[9] = 's';

    if (stmode & S_IRUSR)
        pMode[1] = 'r';
    if (stmode & S_IWUSR)
        pMode[2] = 'w';
    if (stmode & S_IXUSR)
        pMode[3] = 'x';

    if (stmode & S_IRGRP)
        pMode[4] = 'r';
    if (stmode & S_IWGRP)
        pMode[5] = 'w';
    if (stmode & S_IXGRP)
        pMode[6] = 'x';

    if (stmode & S_IROTH)
        pMode[7] = 'r';
    if (stmode & S_IWOTH)
        pMode[8] = 'w';
    if (stmode & S_IXOTH)
        pMode[9] = 'x';

    return 0;
}

/* 입력받은 스트링을 정렬하는 함수다. */
int SortingStr(char **str, int sumfiles)
{
    int i, j;
    char tmp[MAX_SIZE];

    for (i = sumfiles - 2; i >= 0; i--)
    {
        for (j = 0; j <= i; j++)
        {

```

```

        if (strcmp(str[j], str[j + 1]) > 0)
        {
            strcpy(tmp, str[j]);
            strcpy(str[j], str[j + 1]);
            strcpy(str[j + 1], tmp);
        }
    }
}

return 0;
}

/*
 *   디렉토리 파일리스트와 총 갯수를 입력받으면
 *   리스트의 내용을 포맷에 맞추어 출력한다.
 */
void EntryPrint(char **filelist, int sumfiles)
{
    struct stat buf;          /* 파일의 상태 저장 버퍼 */
    char tBuf[TIME_SIZE];    /* 시간정보를 포맷에 맞추어 저장할 버퍼 */
    struct tm *tm;            /* 시간정보를 저장할 버퍼 */
    char pMode[CHAR_PERM_SIZE]; /* 퍼미션 모드 스트링 */
    struct passwd *uName;     /* 패스워드 파일을 참조할 구조체 */
    struct group *gName;      /* 그룹정보를 참조할 구조체 */
    char file[MAX_SIZE];      /* 입력받은 엔트리 하나를 임시로 저장할 스트링 변수 */
    int i;

    /*   입력받은 엔트리의 갯수만큼 반복적으로
    각 엔트리의 정보를 flag에 맞게 입력받아 출력한다. */
    for(i=0; i<sumfiles; i++)
    {
        /* 정렬의 순서를 역방향으로 할때 뒤의 엔트리부터 검사한다. */
        if(rFlag)
            strcpy(file, filelist[sumfiles-1-i]);
        else
            strcpy(file, filelist[i]);

        /* 경로안에 모든 내용을 출력하지 않을 때는 "."과 ".."은 제외한다. */
        if(!aFlag && ((strcmp(file, ".")==0) || (strcmp(file, "..")==0)) )
            continue;

        /* 긴 리스트의 포맷으로 출력할때 */
        else if(lFlag)
        {
            /* 파일의 상태를 불러온다 */
            if (stat(file, &buf) == -1)
            {
                perror("ls : \n");
            }
        }
    }
}

```

```

        exit(1);
    }
    tm = localtime(&buf.st_mtime);    /* 파일의 수정시간을 불러온다. */
    /* 불러온 시간을 포맷에 맞추어 저장한다. */
    strftime(tBuf, sizeof(tBuf), outpT, tm);

    memset(pMode, '-', 9 * sizeof(char));
    pMode[10] = '\0';

    StrMode(pMode, buf.st_mode);    /* 파일의 퍼미션 모드를
                                    입력하여 스트링으로 반환받는 함수다. */

    uName = getpwuid(buf.st_uid);    /* 사용자의 User Name */
    gName = getgrgid(buf.st_gid);    /* 사용자의 Group Name */

    printf("\n%s %2d %3s %3s %6d %s %s"
        , pMode, (int)buf.st_nlink, uName->pw_name, gName->gr_name
        , (int)buf.st_size, tBuf, file);    /* 포맷에 맞추어 출력한다. */
}

/* 리스트 포맷이 아니라면 파일 이름만 출력한다. */
else
{
    printf("%s ", file);
}
}
printf("\n\n");
}

/*
 *   디렉토리를 인자로 입력받으면 해당 디렉토리의 엔트리를 검사하여
 *   디렉토리가 존재한다면 같은함수 recursive하게 호출되는 함수다.
 */
int EntrySearch(char *dirpath)
{
    DIR *dir;    /* 입력받은 디렉토리의 정보를 담는다. */
    struct dirent *d; /* 디렉토리의 엔트리 정보를 입력받는다. */
    struct stat buf; /* 파일 정보를 저장한다. */
    int sumfiles = 0, i;    /* 디렉토리 내부의 엔트리 총 갯수다. */
    char **filelist; /* 엔트리의 이름이 스트링들로 저장된다. */
    char absoultepath[MAX_SIZE]; /* 파일의 절대경로를 저장한다. */

    /* 입력받은 인자 디렉토리를 open한다. */
    if ( (dir=opendir(dirpath)) == NULL )
    {
        perror("ls : opendir");
    }

```



```

        exit(1);
    }

    /* 엔트리를 입력받을 메모리를 할당한다. */
    if ( (filelist = (char**)malloc(sizeof(char*) * MAX_SIZE) ) == NULL)
    {
        perror("ls : mem faild\n");
        exit(1);
    }

    /* open된 디렉토리의 엔트리를 읽는다.*/
    while ( (d = readdir(dir)) )
    {
        /* 엔트리를 저장할 메모리를 할당한다. */
        if ((filelist[sumfiles] = (char*)malloc(sizeof(char) * MAX_SIZE) ) == NULL)
        {
            perror("ls : mem faild\n");
            exit(1);
        }
        strcpy(filelist[sumfiles], d->d_name); /* 엔트리 이름을 할당된 메모리에 복사한다.*/
        sumfiles++; /* 엔트리 갯수를 하나 추가한다. */
    }
    closedir(dir); /* 디렉토리를 닫아준다.*/

    SortingStr(filelist, sumfiles); /* 입력받은 엔트리들을 알파벳순으로 정렬하는 함수다.*/
    EntryPrint(filelist, sumfiles); /* 입력받은 엔트리들을 출력하는 함수다. */

    /* 하위디렉토리모두 출력을 한다면 */
    if(RFlag)
        /* 일일이 모든 엔트리를 검사하여 '.'와'..'가 아니고 디렉토리라면 디렉토리의
        절대경로를 출력한 후 엔트리 검사 함수를 recursive하게 호출하여 검사한다. */
        for(i=0;i<sumfiles;i++)
        {
            sprintf(absoultepath,"%s/%s",dirpath,filelist[i]);
            if((strcmp(filelist[i],".")!=0) && (strcmp(filelist[i],"..")!=0)
            && (stat(absoultepath,&buf) == 0))
            {
                if(S_ISDIR(buf.st_mode))
                {
                    printf("%s : \n",absoultepath);
                    EntrySearch(absoultepath);
                }
            }
        }

    /* 동적 메모리를 모두 해제한다.*/
    for(i=0;i<sumfiles;i++)
        free(filelist[i]);
    free(filelist);
    return 0;

```

```

}

/*
 *      LS 명령어
 */
int main(int argc, char **argv)
{
    char *optionTmp;      /* 옵션스트링을 임시로 저장한다. */
    int numfile;          /* 입력받은 인자의 어느 부분부터 탐색하고자 하는
                           위치인지 인덱스를 알려준다 */
    int noin = FALSE;     /* 전달받은 인자가 있다면 현재위치를 탐색한다. */
    char *curD = getcwd(NULL, BUFSIZ); /* 현재의 디렉토리를 받는다. */

    /* 옵션의 존재를 검사한다. */
    if( argc > 1 )
    {
        optionTmp = argv[1];
        if (optionTmp[0] == '-')
        {
            /* 'a' Option */
            if (strchr(optionTmp, A)) aFlag = TRUE;
            /* 'l' Option */
            if (strchr(optionTmp, L)) lFlag = TRUE;
            /* 'R' Option */
            if (strchr(optionTmp, bigR)) RFlag = TRUE;
            /* 'r' Option */
            if (strchr(optionTmp, smallR)) rFlag = TRUE;

            numfile = 2;

            /* 옵션만 있고 입력값이 없다면 현재 위치를 탐색한다. */
            if (argc == 2)
            {
                noin = TRUE;
                numfile = 1;
            }
        }
        /* 전달받은 인자들이 존재한다. */
        else
        {
            numfile = 1; /* 입력값은 전달받은 인자의 2번째 인덱스부터. */
        }
    }
    /* 전달받은 인자가 없다면 현재 위치를 탐색한다. */
    else
    {
        numfile = 0;
    }
}

```

```

        noin = TRUE;
    }

    while (numfile < argc)
    {
        /* 전달받은 인자가 없다면 현재 위치를 탐색한다. */
        if (noin)
        {
            /* 하위 디렉토리 출력 포맷이라면 현재 위치도 출력한다. */
            if (RFlag)
                printf("%s : \n", curD);
            EntrySearch(curD);
        }

        else
        {
            /* 하위 디렉토리 출력 포맷이라면 현재 위치도 출력한다. */
            if (RFlag)
                printf("%s : \n", argv[numfile]);
            EntrySearch(argv[numfile]);      /* 엔트리 탐색을 시작한다. */
        }
        numfile++;
    }
    return 0;
}

```

## od.c

```
/*-----  
고급시스템 프로그래밍 : Command [ od ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*   od명령어는 입력받은 파일들을 octal넘버 혹은 아스키로  
*   변환해 출력해주는 명령어다. 입력받은 파일들은 임시파일  
*   한곳에 모두 저장되어 읽을때는 이 파일에서 1바이트로 읽고  
*   모두 읽으면 삭제한다.  
*   만약 아스키모드가 아니라면 2바이트를 출력하므로 1바이트를  
*   우선 읽고 다음 바이트를 읽어 결합하여 출력 한다.  
*/  
  
#include "minishlib.h"  
  
#define LIMIT_LINE 16 /* 출력 가능한 한 라인의 글자 수*/  
#define NL 0x0a      /* NewLine 아스키넘버 */  
#define SPACE 0x20   /* Space 아스키넘버*/  
#define TAB 0x09     /* Tab 아스키넘버 */  
#define ONE_BYTE 8   /* 한 바이트 */  
  
/*  
*   OD 명령어  
*/  
int main(int argc, char **argv)  
{  
    int aFlag=FALSE; /* 아스키 옵션 플래그다 */  
    char *optionTmp = argv[1]; /* 임시로 옵션을 저장하는 변수다 */  
    FILE *in;           /* 파일 구조체*/  
  
    short buf; /* 입력된 바이트를 저장한다. */  
    short buftmp; /* 아스키모드가 아닐 때 입력받은 첫번째 바이트로  
                  다음입력받는 두번째 바이트와 결합되어 출력된다. */  
    short numCount; /* 입력받은 바이트 수를 저장한다. */  
    int count = 0; /* 오프셋 카운트다. */  
  
    char *dump = "oddump.oddump"; /* 임시로 저장될 파일의 이름이다. */  
    int fileNum = 0; /* 다중파일 입력시 입력받은 인자의 인덱스중  
                     어느 곳부터 파일스트림이 저장된 인덱스인지를 알려준다. */  
    int fpRead,fpWrite; /* 읽을 파일과 쓰기를 할 파일의 파일 포인터이다. */
```

```

char charBuf[BUFSIZ]; /* 열린 파일에서 내용을 읽어저장 할 버퍼다. */
int nread;           /* 읽어온 바이트의 수다 */
int redir = FALSE;   /* 아무파일도 입력받지 않는다면
                      redirection된 것이므로 Standard Input에서 입력받는다. */

/* 옵션을 체크한다. */
if(argc > 1)
{
    if(optionTmp[0] == '-')
    {
        /* 'a' Option */
        if(strchr(optionTmp,A)) aFlag=TRUE;

        /* option만 존재하고 input파일이 없다면 redirection된 것이다. */
        if(argc == 2 )
        {
            redir=TRUE;
            fileNum = 1;
            fpRead=0;    /* standard in 읽기*/
        }
        else
            fileNum = 2;
    }
    /* Option이없다. */
    else
    {
        fileNum = 1;
    }
}

/* input파일이 없다면 redirection된 것이다. */
else
{
    redir=TRUE;
    fileNum = 0;
    fpRead=0;    /* standard in 읽기*/
}

/* 읽은 파일들을 임시로 저장할 파일을 생성한다. */
if( (fpWrite = open(dump,O_WRONLY | O_CREAT | O_TRUNC, 0755)) <0 ){
    perror("od:f open");
    exit(1);
}

/* 파일의 시작부터 마지막 파일까지 모두 읽어 임시 파일에 저장한다. */
while(fileNum < argc){
    if(!redir)

```

```

        /* 전달받은 인자(파일)를 open한다. */
        if( (fpRead = open(argv[fileNum],O_RDONLY)) < 0)
        {
            perror("od: f open");
            return 1;
        }
        /* open한 파일의 내용을 읽어
        버퍼에 저장하고 임시파일에 쓴다.*/
        while( (nread=read(fpRead,charBuf,BUFSIZ))>0 ){
            if(write(fpWrite,charBuf, nread)< nread)
            {
                close(fpRead);
                close(fpWrite);
                return (-3);
            }
        }

        fileNum++;      /* 다음 파일을 가리킨다. */
        close(fpRead);  /* open한 파일을 닫아준다. */
    }
    close(fpWrite);    /* 더이상 임시파일에 쓰지않고 닫아준다. */

    /* 이번에는 임시일을 읽기모드로 file open 한다.*/
    if( (in=fopen(dump,"rb") ) == NULL)
    {
        fputs("[ERROR]File open!\n",stderr);
        exit(1);
    }

    /* 아스키 옵션 모드라면 */
    if(aFlag)
    {
        /* 임시 파일에서 1바이트단위로 읽어 저장한다. */
        while ( (numCount=fread(&buf, sizeof(char),1,in) ) > 0)
        {
            /* 라인의 첫번째 출력 바이트라면 총 갯수를 출력해준다.*/
            if(count % LIMIT_LINE == 0)
                printf("%07o", count);

            /* ASCII '\n' */
            if(buf == NL)
                printf("  nl");

            /* ASCII ' ' */
            else if(buf == SPACE)
                printf("  sp");
        }
    }

```

```

        /* ASCII 'tab' */
        else if(buf == TAB)
            printf("  ht");

        /* ASCII 'others' */
        else
            printf("   %c", buf);

        /* 라인의 마지막 바이트라면 다음줄로 이동한다. */
        if( (count % LIMIT_LINE) == (LIMIT_LINE-1) )
            printf("\n");

        /* 읽을 바이트를 카운팅 해준다. */
        count+=numCount;
    }
}

/* 아스키 모드옵션이 아니라면 */
else
{
    /* 임시 파일에서 1바이트단위로 읽어 저장한다. */
    while ( (numCount=fread(&buf, sizeof(char),1,in) ) > 0)
    {
        /* 라인의 첫번째 출력이라면 총 갯수를 출력한다.*/
        if(count % LIMIT_LINE == 0)
            printf("%07o", count);

        /* 읽어온 바이트의 갯수를 카운팅 한다. */
        count+=numCount;

        /* 출력할 2바이트중 처음 바이트를 읽어와 임시로 저장한다.*/
        if(count % 2 == 1)
            buftmp=buf;

        /* 출력할 2바이트중 두번째 바이트를 읽어와 1바이트 Left쉬프트를 하고
        첫번째 바이트와 OR연산을 통해서 합치고 출력한다. */
        else
        {
            buftmp |= buf << ONE_BYTE;
            printf(" %06o", buftmp);
        }

        /* 1줄의 마지막 이라면 다음줄로 이동한다. */
        if(count % LIMIT_LINE == 0)
            printf("\n");
    }
}

```

```

        /* 마지막에 출력할 2바이트가 1바이트 뿐이라면 쉬프트없이 그냥 출력한다. */
        if( count % 2 == 1)
            printf(" %06o", buf);
    }
    /* 마지막 줄에는 총읽은 바이트 갯수를 octal로 출력한다. */
    printf("\n%07o\n", count);

    /* 임시파일을 삭제한다. */
    if(unlink(dump) < 0)
    {
        perror("od: Unlink");
        exit(1);
    }

    fclose(in);      /* 파일 구조체를 닫아준다. */

    return 0;
}

```



## cat.c

```
/*-----
고급시스템 프로그래밍 : Command [ cat ]

                                국민대학교 컴퓨터공학과
                                20093284 나홍철
-----*/

/*
 *      cat 명령어는 입력받은 파일들을 출력하는 명령어다.
 *      -b옵션은 공백을 제외한 스트링을 라인넘버와 함께 출력되며
 *      -n옵션은 공백을 포함한 스트링을 라인넘버와 함께 출력한다.
 */

#include "minishlib.h"

#define NL '\n'

/*
 *      cat 명령어
 */
int main(int argc, char **argv)
{
    char *optionTmp = argv[1];      /* 임시로 옵션을 저장한다. */
    int fileNum=0;    /* 여러개의 파일을 입력받을 경우 전달받은 인자에서 파일의 시작인덱스다. */
    char charBuf[1];    /* 출력될 값을 한 바이트씩 읽어온다. */
    int nread, bNread;    /* 옵션에 사용된다. '\n' 하나뿐인 라인처리를 위함이다. */
    int bFlag = FALSE;    /* 플래그 옵션이다. bFlag -> 공백을 제외 한 각문장 앞에
                           번호를 표시한다. */
    int nFlag = FALSE;    /* 플래그 옵션이다. nFlag -> 공백을 포함 한 각 문장
                           앞에 번호를 표시한다. */
    int redir = FALSE;    /* 인자에 입력받은 파일이 모두 없다면 redirection 된 것이다. */
    int fpRead;    /* 읽을 파일의 fd 번호다. */
    int count;    /* 라인넘버를 저장한다. */
    char str[BUFSIZ];    /*한 바이트로 입력받은 캐릭터들이 저장되 스트링이 되고 출력된다.*/

    /* 옵션을 체크한다. */
    if(argc > 1)
    {
        if(optionTmp[0] == '-')
        {
            /* 'b' Option */
            if(strchr(optionTmp,B)) bFlag=TRUE;
```

```

        /* 'n' Option */
        else if(strchr(optionTmp,N)) nFlag=TRUE;
        /* 입력파일이없으므로 redirection 된것이다. */
        if(argc == 2)
        {
            redir=TRUE;
            fileNum = 1;
            fpRead=0;      /* standard in */
        }
        /* 입력 파일이 존재한다. */
        else
            fileNum = 2;
    }
    /* 옵션이 존재 하지 않는다. */
    else
    {
        fileNum = 1;
    }
}
/* 옵션과 입력파일이 존재하지 않는다. */
else
{
    redir=TRUE;
    fileNum = 0;
    fpRead=0;      /* standard in */
}

/* 입력받은 입력파일 모두 수행한다. */
while(fileNum<argc)
{
    /* redirection이 아닌 입력파일이 있을 경우 읽기 모드로 파일을 연다. */
    if (!redir)
        if( (fpRead = open(argv[fileNum],O_RDONLY)) < 0)
        {
            perror("cat : f open");
            return 1;
        }

    /* 옵션이 있을 경우 출력을 위한 라인넘버를 출력될 값 앞에 붙여준다. */
    if(bFlag || nFlag)
    {
        count=1;
        sprintf(str,"%d ",count);
    }

    bNread = 0; /* 입력받은 바이트 수를 저장하며 한줄에 '\n'만 있는가를 판별한다. */
}

```

```

/* 열기되거나 혹은 redirection된 파일에서 1바이트씩 읽어온다. */
while( (nread=read(fpRead,charBuf,1))>0 ){
    bNread+=nread;

    /* b옵션이 설정되어 있고 '\n'이 한줄의 전부라면 '\n'만 출력한다.
    b옵션의 경우 공백을 제외하고 출력하므로 다음 줄 처음 시작에 카운트를 붙여준다.*/
    if (bFlag && (charBuf[0]==NL) && bNread==1)
    {
        printf("\t\n");
        sprintf(str,"%d ",count);
        bNread=0;
        continue;
    }

    /* 한줄의 끝에 온다면 출력을 한다. */
    else if (charBuf[0]==NL)
    {
        /* 옵션이 설정되어있다면 카운트가 붙여진 입력받은 스트링을
        출력하고 다음 입력받을 스트링에 카운트를 붙인다. */
        if(bFlag || nFlag)
        {
            printf("\t%s\n",str);
            count++;
            sprintf(str,"%d ",count);
            bNread=0;
        }
        /* 아무 옵션이 없는 경우 카운트는 붙여있지 않으며
        단지 스트링을 출력한다.*/
        else
        {
            printf("%s\n",str);
            sprintf(str,"%c ",0);
        }
    }
    /* 끝이 아니라면 캐릭터를 스트링에 붙인다. */
    else
        sprintf(str,"%s%c",str,charBuf[0]);
}
close(fpRead);

/* 다음 입력파일을 가져온다. */
fileNum++;
}
return 0;
}

```

## touch.c

```
/*-----
고급시스템 프로그래밍 : Command [ touch ]

                                국민대학교 컴퓨터공학과
                                20093284 나홍철
-----*/

/*
 *      touch명령어는 파일이 존재하지 않는다면 생성하고,
 *      존재한다면 기존파일의 변경시간을 변경하는 명령어다.
 *      t옵션이 주어진다면 주어진 시간으로 시간을 변경한다.
 */
#include "minishlib.h"

/*
 *      TOUCH 명령어
 */
int main(int argc, char **argv)
{
    char *optionTmp = argv[1];      /* 임시로 옵션을 받는다. */
    char *ptr;                      /* strtol시 변환되자 않는 스트링위치를 받는 곳 */
    int tFlag = FALSE;             /* time Flag로 touch시 입력받은 시간으로 변경할 때 사용한다. */
    int filenum = 1; /* 받은 인자들중 만들려는 파일의 시작위치 인덱스다. */
    struct utimbuf ubuf;           /* 시간을 변경하기위한 시간 구조체다. */
    int i;
    int t; /* 입력받은 인자중 변경할 시간스트링을 정수형으로 변환 */
    int fd; /* 파일의 존재여부를 open으로 확인한다. */

    /* 옵션을 체크한다. */
    if( argc > 1 )
    {
        if(optionTmp[0] == '-')
        {
            /* 't' Option */
            if(strchr(optionTmp,t))
            {
                tFlag=TRUE;
                filenum = 3; /* 받은 인자중 파일의 시작위치 인덱스다. */
                t = strtol(argv[2], &ptr, 10); /* 입력받은 시간 스트링을
                                                정수형으로 변경해준다.*/
                ubuf.actime = (time_t)t; /* 접근시간을 변경해준다.*/
                ubuf.modtime = (time_t)t; /* 변경시간을 변경해준다. */
            }
        }
    }
}
```

```

        /* 잘못된 옵션이라면 에러처리 */
        else
        {
            printf("touch: Wrong Option\n");
            return 1;
        }
    }

    /* 입력받은 파일들이 존재하지않는다면 생성하고 시간을 변경해준다.*/
    for(i = filenum; i<=argc; i++)
    {
        /* 없는 파일이라면 생성하고 있는 파일이라면 그냥 무시한다. */
        fd = open(argv[i], O_RDONLY | O_CREAT | O_EXCL, 0644);

        /* 시간을 변경하는 옵션이 주어지면 변경한 ubuf시간 값으로
        파일의 inode시간정보를 수정한다. */
        if(tFlag)
            utime(argv[i], &ubuf);
        /* 시간을 변경하는 옵션이 주어지지 않으면 현재의 시간으로 변경한다. */
        else
            utime(argv[i], NULL);
        close(fd);
    }
}

/* 단순히 명령어 하나만 인자로 전달받으면 에러다. */
else
{
    printf("touch : Wrong Input\n");
    return 1;
}

return 0;
}

```

## head (python)

```
#!/usr/bin/python
import sys
import re
import os

def HEAD (arg,argc):
    begin = 0 # Head Command / Begin Point
    fileName = arg[1:]
    nFlag = 0 # Option : including file 'name Flag'
    lFlag = 0 # Option : limit num of 'line Flag'
    redir = 0 # redirection

    if argc > 1:
        # Option
        if arg[1][0] == '-' :
            fileName = arg[2:] # File Names except Option
            # verbose option
            if bool(re.search('v',arg[1])) :
                nFlag = 1
            # quite option
            if bool(re.search('q',arg[1])) :
                nFlag = 0
            # lines option
            if bool(re.search('n',arg[1])) :
                try:
                    if int(arg[2]) >= 0:
                        lFlag = 1
                        limitLines = int(arg[2])
                        fileName = arg[3:] #File Names except Num

                        if argc == 3:
                            redir=1

                except:
                    print "[ERROR] Wrong number of lines"
                    sys.exit()

            # No input file -> standard in
            if argc == 2 and ~lFlag:
                redir=1

    else :
        redir=1

    # redirection
```

```

if redir :
    line = sys.stdin.readlines()
    try:
        # End point of printing
        if lFlag :
            end=limitLines
        else :
            end=len(line)

        # File Name
        if nFlag :
            print "==> Standard Input <=="

        for i in range(begin,end):
            print re.sub("\n","",line[i]) # '\n' Delete line

    except :
        print "head : cannot open standard in!\n"

# input files
else :
    for n in fileName :
        try:
            if os.path.exists(n) :
                fr = open(n, 'r')
                line = fr.readlines()

                # End point of printing
                if lFlag :
                    end=limitLines
                else :
                    end=len(line)

                # File Name
                if nFlag :
                    print "==> "+n+" <=="

                for i in range(begin,end):
                    print re.sub("\n","",line[i]) # '\n' Delete line

            except :
                print "[ERROR] cannot open \" + n + "\" for reading: No such
file or directory!\n"

HEAD(sys.argv,len(sys.argv))

```

## tail (python)

```
#!/usr/bin/python
import sys
import re
import os
def TAIL (arg,argc):
    begin = 0 # Head Command / Begin Point
    fileName = arg[1:]
    nFlag = 0 # Option : including file name Flag
    lFlag = 0 # Option : limit num of line Flag
    redir = 0 # redirection

    if argc > 1:
        # Option
        if arg[1][0] == '-' :
            fileName = arg[2:] # File Names except Option
            # verbose option
            if bool(re.search('v',arg[1])) :
                nFlag = 1
            # quite option
            if bool(re.search('q',arg[1])) :
                nFlag = 0
            # lines option
            if bool(re.search('n',arg[1])) :
                try:
                    if int(arg[2]) >= 0:
                        lFlag = 1
                        limitLines = int(arg[2])
                        fileName = arg[3:] #File Names except Num
                        if argc == 3:
                            redir=1

                except:
                    print "[ERROR] Wrong number of lines"
                    sys.exit()

            # No input file -> standard in
            if argc == 2 and ~lFlag:
                redir=1

    else :
        redir=1

    # redirection
```



```

if redir :
    line = sys.stdin.readlines()
    try:
        # End point of printing
        if lFlag :
            begin=len(line)-limitLines
            end=len(line)
        else :
            end=len(line)

        # File Name
        if nFlag :
            print "==> Standard Input <=="

        for i in range(begin,end):
            print re.sub("\n","",line[i]) # '\n' Delete line
    except :
        print "head : cannot open standard in!\n"

# input files
else :
    for n in fileName :
        try:
            if os.path.exists(n) :
                fr = open(n, 'r')
                line = fr.readlines()

                # End point of printing
                if lFlag :
                    begin=len(line)-limitLines
                    end=len(line)
                else :
                    end=len(line)

                # File Name
                if nFlag :
                    print "==> "+n+" <=="

                for i in range(begin,end):
                    print re.sub("\n","",line[i]) # '\n' Delete line

        except :
            print "[ERROR] cannot open '\" + n + '\" for reading: No such
file or directory!\n"

TAIL(sys.argv,len(sys.argv))

```

## chmod.c

```
/*-----  
고급시스템 프로그래밍 : Command [ chmod ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*      chmod 명령어로 해당 파일과 디렉토리의  
*      퍼미션 모드를 변경해 줄수 있다.  
*      이때 퍼미션모드는 숫자와 문자로 변경이 가능하다.  
*      문자로 입력받는 다면 그 문자 스트링을 변경해주는  
*      함수를 통해 넘버 퍼미션으로 변경한다.  
*/  
#include "minishlib.h"  
  
/* 스트링을 정수형 퍼미션 모드로 변경해준다. */  
int StrChange(char *modeStr, int modeNum)  
{  
    int operFlag;    // '-' => 0 / '+' => 1 / '=' => 2  
    int shiftNum;    /* 넘버 퍼미션에서 각각 user는 6비트, other는 3비트 왼쪽으로 이동한다.*/  
    int i,tmp;  
  
    for( i = 0 ; i<strlen(modeStr);i++)  
    {  
        tmp=0;  
        /* 넘버 퍼미션에서 얼마나 쉬프트를 해야하는지 정한다.  
        총 9비트의 user, group, other 비트를 말하는것임. */  
        if(modeStr[i] == 'u')  
            shiftNum=6;  
        else if(modeStr[i] == 'g')  
            shiftNum=3;  
        else if(modeStr[i] == 'o')  
            shiftNum=0;  
        else  
        {  
            printf("chmod: Wrong Command [u,g,o]\n");  
            return 1;  
        }  
  
        i++;  
        /* '-'는 기본 퍼미션에서 제거  
        '+'는 기본 퍼미션에 추가
```

```

'='는 기본 퍼미션을 무시하고 덮어쓰기가 된다. */
if(modeStr[i]=='-')
    operFlag = 0;
else if(modeStr[i]=='+')
    operFlag = 1;
else if(modeStr[i]=='=')
    operFlag = 2;
else
{
    printf("chmod: Wrong Operator\n");
    return 1;
}

i++;
/* 스트링에서 read, write, excute를 찾는다. */
while(i<strlen(modeStr) )
{
    if(modeStr[i]=='r')
    {
        tmp=tmp|4;
        i++;
    }
    else if(modeStr[i]=='w')
    {
        tmp=tmp|2;
        i++;
    }
    else if(modeStr[i]=='x')
    {
        tmp=tmp|1;
        i++;
    }
    else if(modeStr[i]==',')
        break; /* 다음 모드 변경 스트링값을 찾는다. */
    else
    {
        printf("chmod: Wrong Command\n");
        return 1;
    }
}
/*분석된 스트링의 모드 변경값을 실제 파일의 모드와 함께 알맞게 처리한다. */
/* '-' */
if(operFlag == 0)
    modeNum = (modeNum & ~(tmp << shiftNum));
/* '+' */
else if(operFlag == 1)
    modeNum = (modeNum | (tmp << shiftNum));

```

```

        /* '=' */
        else if(operFlag == 2)
        {
            /* 새로 덮어쓸 부분을 깨끗하게 지운다. */
            if (shiftNum == 6)
                modeNum=modeNum & 65087; // 177077 (octal)
            else if (shiftNum == 3)
                modeNum=modeNum & 65479; // 177707 (octal)
            else if (shiftNum == 0)
                modeNum=modeNum & 65528; // 177770 (octal)

            modeNum = (modeNum | (tmp << shiftNum) );
        }
    }
    return modeNum; /* 넘퍼 퍼미션 모드를 리턴한다. */
}

/*
 * chmod 명령어
 */
int main(int argc, char **argv)
{
    char *modeTmp = argv[1]; /* 임시로 모드라고 지정한다. */
    char *name; /* 변경할 파일,폴더의 이름 */
    char *ptr; /* 토큰 포인터다 */
    int modeNum; /* 변경할 퍼미션 넘버*/
    struct stat buf;

    /* 인자를 체크한다. */
    if(argc > 2)
    {
        name = argv[2];
        /* 만약 임시로 모드라고 지정한 값이 숫자라면 퍼미션 모드는 숫자 */
        if(isdigit(modeTmp[0]))
        {
            modeNum = strtol(modeTmp, &ptr, 8); /*퍼미션숫자를 8진수로 변경한다. */

            /* 입력받은 넘버로 퍼미션모드를 바꾸어 준다 */
            if( chmod(name, modeNum)<-1)
                perror("chmod: Call to chmod failed.");
        }
        /* 퍼미션 모드가 문자 */
        else
        {
            /* 파일이 존재 하지 않는다면 에러*/
            if (stat(name,&buf) == -1)

```

```

        {
            printf("chmod: \n");
            exit(1);
        }
        modeNum = buf.st_mode; /* 해당 파일의 퍼미션 모드를 할당받는다. */

        /* 퍼미션 모드를 변경한다. 이때 문자퍼미션 입력시
        퍼미션 넘버를 반환해주는 함수를 사용하여 반환받아 사용한다. */
        if( chmod(name, StrChange(modeTmp,modeNum))<-1)
            perror("chmod: Call to chmod failed.");
    }
}
else
{
    printf("chmod:Wrong Command\n");
    return 1;
}

return 0;
}

```

## cd.c

```

/**
 * [ Change Directory ] Shell 프로그램에 입력되어있는 함수다.
 * 자식 프로세서에서 실행 하여서는 안되며 Shell에서 실행되어야 디렉토리 변경이 가능하다.
 */
int CDcommand(int argc, char **argv)
{
    char *chpath=getenv("HOME"); /* HOME 환경변수인 홈 디렉토리를 받아둔다. */

    /* 넘겨받은 인자가 있다면 받은 위치의 절대경로를 취해주어 해당 위치로 변경한다. */
    if(argc > 0)
        realpath(argv[1],chpath);
    chdir(chpath);
    return 0;
}

```

## **pwd.c**

```
/*-----  
고급시스템 프로그래밍 : Command [ pwd ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*      pwd명령어는 Current directory를 출력한다.  
*/  
  
#include "minishlib.h"  
  
/*  
*      PWD 명령어  
*/  
int main()  
{  
    char *curD;  
  
    curD = getcwd(NULL, BUFSIZ); /* Currnet Directory를 저장한다. */  
    printf("Current Directory : %s\n", curD);  
  
    return 0;  
}
```

## cp.c

```
/*-----  
고급시스템 프로그래밍 : Command [ cp ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*      cp 명령어는 파일 혹은 디렉토리를 복사하는 명령어다.  
*      -f 옵션 : 강제로 복사한다.  
*      -R 옵션 : 디렉토리 내용들 모두 복사한다.  
*/  
#include "minishlib.h"  
  
int fFlag=FALSE, RFlag=FALSE;  
  
/* 파일을 입력받은 file1위치의 파일을 file2에 복사한다. */  
int CopyFile(char *file1, char *file2)  
{  
    int fpRead, fpWrite;  
    char charBuf[BUFSIZ];  
    int nread;  
  
    // reading file fd  
    if ((fpRead = open(file1, O_RDONLY)) < 0)  
    {  
        perror("cp: copy read file open");  
        exit(-3);  
    }  
  
    // writing file fd  
    if ((fpWrite = open(file2, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0){  
        perror("cp: copy write file open");  
        exit(-3);  
    }  
  
    // Copy reading file to writing file  
    while ((nread = read(fpRead, charBuf, BUFSIZ))>0)  
    if (write(fpWrite, charBuf, nread)< nread)  
    {  
        close(fpRead);  
        close(fpWrite);  
        exit(1);  
    }  
}
```

```

    }
    close(fpRead);
    close(fpWrite);

    return 0;
}

/* 디렉토리를 복사하는 함수다. */
int CopyDir(char *fromDir, char *toDir)
{
    DIR *dir;
    struct dirent *fromd;
    struct stat buf;
    char fromAbsPath[MAX_SIZE], toAbsPath[MAX_SIZE];
    int mkdErr=0;
    char ans[MAX_SIZE];

    /* 복사 될 곳의 디렉토리를 생성한다 만약 존재하는 에러 이외의 에러는 에러다.*/
    if(mkdir(toDir, 0755) < 0 && errno != EEXIST )
    {
        perror("cp: mkdir error");
        return 1;
    }

    /* 전달받은 복사 할 디렉토리를 open한다. */
    if( (dir=opendir(fromDir)) == NULL )
    {
        perror("cp: opendir");
        exit(1);
    }

    /* 복사 할 디렉토리의 엔트리를 읽어 복사될곳에 복사해주기위한 작업을 한다. */
    while ( (fromd = readdir(dir)) != NULL )
    {
        if((strcmp(fromd->d_name, ".")==0) || (strcmp(fromd->d_name, "..")==0) )
            continue;

        sprintf(toAbsPath, "%s/%s", toDir, fromd->d_name);
        sprintf(fromAbsPath, "%s/%s", fromDir, fromd->d_name);

        /* 복사 할 파일의 상태를 확인한다. */
        if (stat(fromAbsPath, &buf) < 0)
        {
            perror("cp : Error Stat");
            continue;
        }

        /* 만약 디렉토리라면 디렉토리 복사 함수를 재귀적으로 호출한다. */
    }
}

```



```

        if(S_ISDIR(buf.st_mode))
        {
            CopyDir(fromAbsPath,toAbsPath);
            continue;
        }

        /* 디렉토리가 아닌 파일이면서 복사될 곳에 이미 같은 이름이 존재한다면
        복사할 지를 물어본다. */
        else if(!fFlag && (stat(toAbsPath,&buf) == 0 ))
        {
            printf("cp: overwrite '%s'? [y/n]",toAbsPath);
            fgets(ans, MAX_SIZE, stdin); /* 스트링을 입력 받는다. */

            /* 붙여넣기 하지 않는다. */
            if(strcmp(ans,YES) != 0)
            {
                continue;
            }
        }
        CopyFile(fromAbsPath, toAbsPath);          /* 파일 복사를 진행한다. */
    }
    closedir(dir);
    return 0;
}

/*
 * cp 명령어*/
int main(int argc, char **argv)
{
    char *optionTmp = argv[1]; // Temporally get option
    char *name1,*name2;
    char n1RealPath[MAX_SIZE],n2RealPath[MAX_SIZE],n2tmpPath[MAX_SIZE];
    int name2Exist=1;
    char *ptr,*name1Ptr; // For Name Split
    struct stat buf1,buf2;
    char charBuf[BUFSIZ];
    char ans[MAX_SIZE];
    int beginFile;

    /* 옵션을 체크한다. */
    if(argc > 2)
    {
        if(optionTmp[0] == '-')
        {
            beginFile = 2;
            name1 = argv[beginFile];
            name2 = argv[argc-1];

```

```

        /* 'f' Option */
        if(strchr(optionTmp,F))    fFlag=TRUE;
        /* 'R' Option */
        if(strchr(optionTmp,bigR)) RFlag=TRUE;
    }
    /* 옵션이 존재하지 않는다면 시작파일은 넘겨받은 인자중
    명령어 다음이며 여러개의 파일 인자값들이 존재할 때는
    마지막 인자는 디렉토리이어야 한다. */
    else
    {
        beginFile = 1;
        name1 = argv[beginFile];
        name2 = argv[argc-1];
    }
}
else
{
    printf("cp : Wrong Command\n");
    return 1;
}

/* 단 두개의 인자만 존재한다면 한파일을 다른 파일로 복사하는것이지만
여러개의 파일이 존재하는 경우 여러인자 파일들이 마지막 한 디렉토리에 복사되는 것이다. */
while( strcmp(name1,name2) != 0)
{
    realpath(name1, n1RealPath);          /* 절대경로로 치환한다.*/
    realpath(name2, n2RealPath);          /* 절대경로로 치환한다.*/

    if (stat(n1RealPath, &buf1) < 0) {
        printf("cp : name1 not exist\n");
        return 1;
    }
    if (stat(n2RealPath, &buf2) < 0) {
        name2Exist = 0; /* name2 not exist */
    }

    /* Name1 is directory */
    if (S_ISDIR(buf1.st_mode))
    {
        /* Directory Copy option true */
        if(RFlag)
        {
            /* Name2 is not directory */
            if (!S_ISDIR(buf2.st_mode))
            {
                printf("cp : cannot overwrite non-directory\n");
                return 1;
            }
        }
    }
}

```

```

        }
        /* Name2 is directory or not exist. */
        else
        {
            /* 디렉토리 복사 함수를 수행한다. */
            CopyDir(n1RealPath, n2RealPath);
        }
    }
    /* Directory Copy option false */
    else
    {
        printf("cp : omitting directory '%s'\n",name1);

        return 1;
    }
}

/* Name1 is not directory */
else
{
    /* Name2 is directory => renew dir(name2) to file(name2/name1) */
    if (S_ISDIR(buf2.st_mode))
    {
        realpath(name1,n1RealPath);

        /* split just name */
        /* Split name1 -> Just File name */
        ptr = strtok( name1 , "/");
        name1Ptr = ptr;
        while(ptr != NULL)
        {
            name1Ptr = ptr;
            ptr = strtok(NULL,"/");
        }
        sprintf(n2tmpPath,"%s/%s",n2RealPath,name1Ptr);
        /* 'n1RealPath' -> 'n1RealPath/name1'*/
        if (stat(n2tmpPath, &buf2) < 0) {
            name2Exist = 0; // name2 not exist
        }

        /* No Force Copy Option && name2 exist file */
        if(!fFlag && name2Exist )
        {
            printf("cp: overwrite '%s'? [y/n]",n2tmpPath);
            fgets(ans, MAX_SIZE, stdin); /* 스트랑 입력. */
            /* 강제 복사 하지 않는다. */

```

```

        if(strcmp(ans,YES) != 0)
        {
            beginFile++;
            name1=argv[beginFile];
            continue;
        }

    }
    CopyFile(n1RealPath, n2tmpPath); /* 파일을 복사한다. */

}
/* name1 is file name2 is file */
else{
    if( ((argc -1) - beginFile) != 1)
    {
        printf("cp : '%s' is not directory\n",name2);
        return 1;
    }

    /* No Force Copy Option && name2 exist file */
    if(!fFlag && name2Exist )
    {
        printf("cp: overwrite '%s'? [y/n]",n2RealPath);
        fgets(ans, MAX_SIZE, stdin); /* 스트림을 입력 받는다. */
        /* 강제 복사 하지않는다. */
        if(strcmp(ans,YES) == 0)
        {
            return 0;
        }
    }
    CopyFile(n1RealPath, n2RealPath); /* 파일을 복사한다. */
    return 0;
}

}

beginFile++; /* 다음 인자를 복사한다. */
name1=argv[beginFile]; /* 다음 인자를 복사한다. */
}
return 0;
}

```

## mv.c

```
/*-----  
고급시스템 프로그래밍 : Command [ mv ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
* cat 명령어는 입력받은 파일들을 출력하는 명령어다.  
* -b 옵션은 공백을 제외한 스트링을 라인넘버와 함께 출력되며  
* -n 옵션은 공백을 포함한 스트링을 라인넘버와 함께 출력한다.  
*/  
  
#include "minishlib.h"  
  
/* 파일을 입력받은 file1 위치의 파일을 file2에 복사한다. */  
int CopyFile(char *file1, char *file2)  
{  
    int fpRead, fpWrite;  
    char charBuf[BUFSIZ];  
    int nread;  
  
    // reading file fd  
    if ((fpRead = open(file1, O_RDONLY)) < 0)  
    {  
        perror("cp: copy read file open");  
        exit(-3);  
    }  
  
    // writing file fd  
    if ((fpWrite = open(file2, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0){  
        perror("cp: copy write file open");  
        exit(-3);  
    }  
  
    // Copy reading file to writing file  
    while ((nread = read(fpRead, charBuf, BUFSIZ)) > 0)  
    if (write(fpWrite, charBuf, nread) < nread)  
    {  
        close(fpRead);  
        close(fpWrite);  
        exit(1);  
    }  
}
```

```

close(fpRead);
close(fpWrite);

return 0;
}

/*
 *   MV 명령어
 */
int main(int argc, char **argv)
{
    int fFlag=FALSE;
    char *optionTmp = argv[1];      /* 임시로 저장해둔 옵션 */
    char absoultepath[BUFSIZ];      /* 절대경로를 저장하는 스트링 */
    char mvfile[BUFSIZ];            /* 옮길 파일이름 */
    char ans[BUFSIZ];               /* 위치에 이미 파일이 존재할 때 덮어쓸지에 대한 답 */
    struct stat buf;               /* 파일 상태를 저장할 구조체 */

    int fbegin, fend;              /* 전달받은 인자들중 파일의 시작과 끝 인덱스 */
    int dirflag = 0, existflag = 0; /* dirflag-> 마지막 인자값이 디렉토리라면 체크,
                                     existflag-> 마지막 인자값이 존재한다면 체크, 덮어쓸지를 물어볼때 사용 */

    fend = argc - 1; /* 전달 받은 인자중 마지막 인덱스 */

    /* 옵션 체크 */
    if(argc > 2)
    {
        if(optionTmp[0] == '-')
        {
            fbegin = 2;
            /* 'f' Option */
            if(strchr(optionTmp,F))
                fFlag=TRUE;
        }
        /* Not Existing Option */
        else
        {
            fbegin = 1;
        }
    }
    else
    {
        printf("mv: Wrong Input\n");
        return 1;
    }
}

```

```

/*last file is existed */
if(stat(argv[fend],&buf) == 0)
{
    /* last file is directory */
    if(S_ISDIR(buf.st_mode))
    {
        existflag=1;
        dirflag=1;
    }

    /* last file is not directory */
    else
    {

        // ex) if "mv file1 file2 file3", but file3 is not directory. file3 is must be existed
directory!

        if(fend - fbegin > 1)
        {
            printf("mv: %s is not directory",argv[fend]);
            return 1;
        }

        existflag=1;
        dirflag=0;
    }
}

/* last file is not existed */
else
{
    // ex) if "mv file1 file2 file3", but file3 is not existed. file3 is must be existed
directory!

    if(fend - fbegin > 1)
    {
        printf("mv: %s is not directory",argv[fend]);
        return 1;
    }
    existflag=0;
    dirflag=0;
}

/* move 'file1' to 'dir/file2' */
if(dirflag)
{
    realpath(argv[fend],absoultepath);
    while (fbegin != fend)
    {

```

```

        sprintf(mvfile, "%s/%s", absoltepath, argv[fbegin]);
        /* Existing 'file2' */
        if( (stat(mvfile,&buf) == 0) && (fFlag==FALSE) )
        {
            printf("mv: overwrite \"%s/%s\"?[y/n] ", argv[fend], argv[fbegin]);
            /* Overwrite? */
            fgets(ans, BUFSIZ, stdin);
            if(strcmp(ans, "y") != 0)
            {
                fbegin++;
                continue;
            }
        }
        /* 이동할 곳에 파일을 복사한다 */
        if (CopyFile(argv[fbegin], mvfile) == 1)
            printf("ERROR : [mv] copy error\n");

        /* 원래 위치의 파일을 지운다. */
        if(unlink(argv[fbegin]) < 0)
        {
            perror("ERROR: [mv] Unlink");
            return 1;
        }
        fbegin++;          /* 다음 파일을 수행한다. */
    }
}

/* move 'file1' to 'file2' */
else
{
    /* Existing 'file2' */
    if(existflag && (fFlag==FALSE) )
    {
        printf("mv: overwrite \"%s\"?[y/n] ", argv[fend]);
        /* Overwrite? */
        fgets(ans, BUFSIZ, stdin);
        if(strcmp(ans, "y") != 0)
        {
            fbegin++;
            return 0;
        }
    }
    /* file1을 file2에 복사한다. */
    if (CopyFile(argv[fbegin], argv[fend]) == 1)
    {
        printf("ERROR : [mv] copy error\n");
    }
}

```



```
        return 1;
    }

    /* file1을 지워 이동한 것처럼 보이게 한다. */
    if(unlink(argv[fbegin]) <0)
    {
        perror("ERROR: [mv] Unlink");
        return 1;
    }
}
return 0;
}
```

## rm.c

```
/*-----  
고급시스템 프로그래밍 : Command [ rm ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*   rm 명령어는 파일 혹은 디렉토리를 제거하는 명령어다.  
*   -f 옵션 : 강제로 제거한다.  
*   -r 옵션 : 디렉토리 내용들 모두 제거한다.  
*/  
  
#include "minishlib.h"  
  
#define IS_DIRECTORY 0  
#define IS_FILE 1  
  
int fFlag=FALSE,rFlag=FALSE;  
  
/* 전달받은 위치의 디렉토리를 제거하는 함수다 */  
int DeleteDir(char *path)  
{  
    DIR *dir;  
    struct dirent *d;  
    struct stat buf;  
    char name[MAX_SIZE]; /* 디렉토리 내부의 엔트리 이름을 절대경로로 저장한다. */  
    char ans[MAX_SIZE];  
  
    /* 전달받은 위치의 디렉토리 경로를 open한다. */  
    if( (dir=opendir(path)) == NULL )  
    {  
        perror("opendir");  
        exit(1);  
    }  
  
    /* 디렉토리 내부 엔트리들을 읽는다. */  
    while ( (d = readdir(dir)) )  
    {  
        /* '.', '..' 은 제외한다. */  
        if((strcmp(d->d_name,".")==0) || (strcmp(d->d_name,"..")==0) )  
            continue;
```

```

sprintf(name, "%s/%s", path, d->d_name); /* 절대 경로로 만든다. */

if (stat(name,&buf) <0)
{
    perror("Error Stat");
    continue;
}
/* 디렉토리라면 디렉토리 제거함수를 재귀적으로 수행한다. */
if(S_ISDIR(buf.st_mode) )
{
    /* 디렉토리 내부를 탐색 제거 할지를 결정한다. */
    if(fflag==FALSE)
    {
        printf("rm: descend into directory '%s'?[y/n] ",name);
        fgets(ans, MAX_SIZE, stdin); /* 스트랑을 입력 받는다. */
        /* 탐색 지우기 원치 않는다. */
        if(strcmp(ans,YES) != 0)
        {
            continue;
        }
    }

    DeleteDir(name); /* 디렉토리 경로로 디렉토리 제거 함수를 재귀적으로 사용*/

    /* 만약 강제로 지우는 'f'옵션이 아니라면 물어본다. */
    if(fflag==FALSE)
    {
        printf("rm: remove '%s'?[y/n] ",name);
        fgets(ans, MAX_SIZE, stdin); /* 스트랑을 입력 받는다. */
        /* 강제로 지우기 원치 않는다. */
        if(strcmp(ans,YES) != 0)
        {
            continue;
        }
    }
    /* 디렉토리를 제거한다.*/
    if(remove(name) <0)
    {
        perror("Unlink Error");
        continue;
    }
}
/* 디렉토리가 아닌 파일이라면 */
else
{
    /* 만약 강제로 지우는 'f'옵션이 아니라면 물어본다. */
    if(fflag==FALSE)

```

```

        {
            printf("rm: remove '%s'?[y/n] ",name);
            fgets(ans, MAX_SIZE, stdin); /* 스트링을 입력 받는다. */
            /* 강제로 지우지 않는다. */
            if(strcmp(ans,YES) != 0)
            {
                continue;
            }
        }

        /* 파일을 제거한다.*/
        if(unlink(name) <0)
        {
            perror("Unlink Error");
            return 1;
        }
    }
}
closedir(dir);
return 0;
}

/*
 *   RM 명령어
 */
int main(int argc, char **argv)
{
    char *optionTmp = argv[1]; /* 옵션이라고 임시로 지정 */
    int fileNum = 0; /* 전달받은 인자중 파일이 어느 인덱스부터 시작인가를 알려준다. */
    struct stat buf;
    char name[MAX_SIZE]; /* */
    char ans[MAX_SIZE];

    /* Option Check */
    if(argc > 1)
    {
        /* Existing Option */
        if(optionTmp[0] == '-')
        {
            fileNum=2;
            /* 'f' Option */
            if(strchr(optionTmp,F)) fFlag=TRUE;
            /* 'r' Option */
            if(strchr(optionTmp,smallR)) rFlag=TRUE;
        }
        /* Not Existing Option */
        else

```

```

        {
            fileNum=1;
        }
    }

    else
    {
        printf("rm: Wrong Command\n");
        return 1;
    }
    while(fileNum < argc)
    {
        realpath(argv[fileNum],name);
        /* Exist File */
        if(stat(name,&buf) == 0 )
        {
            /* name2 = Directory && rFlag option true */
            if(S_ISDIR(buf.st_mode) )
            {
                if(rFlag==TRUE)
                {
                    /*rm: remove regular empty file 'c/file1'? */
                    if(fFlag==FALSE)
                    {
                        printf("rm: descend into directory '%s'?[y/n] ",name);

                        fgets(ans, MAX_SIZE, stdin);/*스트랑을 입력 */
                        /* Not over write */
                        if(strcmp(ans,YES) != 0)
                        {
                            fileNum++;

                            continue;
                        }
                    }
                    DeleteDir(name); /* 디렉토리 제거 함수를 수행한다. */

                    if(fFlag==FALSE)
                    {
                        printf("rm: remove '%s'?[y/n] ",name);
                        fgets(ans, MAX_SIZE, stdin);/*스트랑을 입력 */
                        /* Not over write */
                        if(strcmp(ans,YES) != 0)
                        {
                            fileNum++;

```

```

                                continue;
                                }
                                }
                                /* delete directory */
                                if(remove(name) <0)
                                {
                                    perror("rm : Unlink Error");
                                    continue;
                                }
                                }
                                else
                                {
                                    printf("rm : Can't delete Directory '%s'\n",name);
                                }
                                }
                                /* name == file */
                                else
                                {
                                    if(fFlag==FALSE)
                                    {
                                        printf("rm: remove '%s'?[y/n] ",name);
                                        fgets(ans, MAX_SIZE, stdin); /* 스트랑을 입력 받는다. */
                                        /* Not over write */
                                        if(strcmp(ans,YES) != 0)
                                        {
                                            continue;
                                        }
                                    }
                                    /* 파일을 지운다. */
                                    if(unlink(name) <0)
                                    {
                                        perror("rm : Unlink Error");
                                        return 1;
                                    }
                                }
                                }
                                else
                                {
                                    printf("rm : File Not exist\n");
                                }
                                fileNum++;      /* 다음 파일을 처리한다 */
                                }
                                return 0;
                                }

```

## ln.c

```
/*-----
고급시스템 프로그래밍 : Command [ ln ]

                                국민대학교 컴퓨터공학과
                                20093284 나홍철
-----*/

/*
 *   ln 명령어는 파일혹은 디렉토리의 hard link, soft link를 설정하는
 *   명령어다.
 *   -s 옵션 : soft link를 설정한다.
 */

#include "minishlib.h"

/*
 * LN 명령어
 */
int main(int argc, char **argv)
{
    struct stat buf; // Stat structure
    char *optionTmp = argv[1]; // Temporally get option
    int sFlag = FALSE; // Option 's'(symbolick link) flag
    int i;
    char *name1,*name2; // 링크 할 인자(name1)와 링크 될 인자(name2)

    // Option Check
    if(argc > 2)
    {
        // Existing Option
        if(optionTmp[0] == '-')
        {
            name1 = argv[2];
            name2 = argv[3];
            // 's' Option
            if(strchr(optionTmp,S))    sFlag=TRUE;
        }
        // Not Existing Option
        else
        {
            name1 = argv[1];
            name2 = argv[2];
        }
    }
}
```

```
}  
// Lack of arguments  
else  
{  
    printf("ERROR : [ln] Wrong Command\n");  
    return 1;  
}  
  
// Symbolic Link  
if(sFlag) symlink(name1,name2);  
  
// Hard Link  
else link(name1,name2);  
  
return 0;  
}
```



## mkdir (python)

```
#!/usr/bin/python

import sys
import re
import os

def MKDIR (arg):
    pMode = 0755 # Basic Permission Mode
    pFlag = 0
    dirName = arg[1:] # Directory Names

    # Option
    if arg[1][0] == '-' :
        dirName = arg[2:] # Directory Names except Option
        # mode option
        if bool(re.search('m',arg[1])) :
            try:
                if int(arg[2]) >= 0: # if intager
                    pMode = int(arg[2],8) # input Permission mode
                    dirName = arg[3:] # File Names except Num
            except:
                print "[ERROR] Wrong permission"
                sys.exit()

        # parent option
        if bool(re.search('p',arg[1])):
            pFlag = 1

    for n in dirName :
        if not os.path.isdir(n) :
            if pFlag :
                os.makedirs(n,pMode) # Make Parents Dir
            else :
                os.mkdir(n,pMode)
        else:
            print "[ERROR] cannot create directory \"' + n + \"': File exists"

MKDIR(sys.argv)
```

## who.c

```
/*-----  
고급시스템 프로그래밍 : Command [ who ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
*   who 명령어는 현재 접속되어 있는 user들의 계정명, 터미널정보,  
*   접속시간을 utmp파일을 참조하여 알 수 있다.  
*/  
  
#include "minishlib.h"  
  
/* Time output standard */  
char *outpT = "%F %H:%M";  
  
/*  
*   who 명령어  
*/  
int main(int argc, char **argv)  
{  
    struct utmpx *utx; /* utmp 파일 구조체이다 */  
    struct tm *tm; /* time 구조체 */  
    char tBuf[20]; /* 포맷에 맞추어진 시간을 저장한다. */  
    time_t time; /* 시간 변수 */  
  
    /* utmp 파일에서 읽은 엔트리를 하나하나 처리한다. */  
    while ((utx = getutxent()) != NULL)  
    {  
        /* user process만 처리한다. */  
        if(utx->ut_type != USER_PROCESS)  
            continue;  
        time = (time_t)utx->ut_tv.tv_sec; /* 시간을 처리한다. */  
        tm = localtime(&time); /* 시간정보를 가져온다. */  
        strftime(tBuf, sizeof(tBuf), outpT, tm); /* 포맷에 맞추어 시간을 정의 한다. */  
  
        printf("%s    %-8s    %s (:%d)\n",  
            ,utx->ut_user,utx->ut_line,tBuf,utx->ut_session);  
    }  
    return 0;  
}
```

## find.c

```
/*-----  
고급시스템 프로그래밍 : Command [ find ]  
  
국민대학교 컴퓨터공학과  
20093284 나홍철  
-----*/  
/*  
* find 명령어는 찾고자 하는 파일과 디렉토리를  
* currnet working directory로 부터 찾아준다.  
* -name 옵션사용시 해당하는 이름의 파일및 디렉토리를 찾아준다.  
* -type 옵션사용시 해당하는 타입의 파일및 디렉토리를 찾아준다.  
* -user 옵션사용시 해당하는 user의 파일및 디렉토리를 찾아준다.  
*/  
  
#include "minishlib.h"  
  
#define OPTION_NAME 0  
#define OPTION_USER 1  
#define OPTION_TYPE 2  
  
#define DI 'd'  
#define BL 'b' // Block Device  
#define CH 'c' // Char Device  
#define LN 'l' // Symbol  
#define RE 'r' // Regular  
  
int optionFlag=0; // 0-> name , 1->user , 2->type  
  
/* 디렉토리 내부를 찾아보는 함수다. */  
int FindEntry(char *dirPath, char *findName)  
{  
    DIR *dir;  
    struct dirent *d;  
    struct stat buf;  
    struct passwd *uName; /* password 파일의 내용을 저장할 구조체이다. */  
    char absPath[MAX_SIZE]; /* 절대경로를 저장할 스트링*/  
    char mode;  
  
    /* 전달받은 인자디렉토리를 연다. */  
    if( (dir=opendir(dirPath)) == NULL )  
    {
```

```

        perror("find: opendir");
        exit(1);
    }
    /* 디렉토리 내부 엔트리를 검사한다. */
    while ( (d = readdir(dir)) != NULL )
    {
        /* '.', '..' 는 검색에서 제외한다. */
        if((strcmp(d->d_name,".")==0) || (strcmp(d->d_name,"..")==0) )
            continue;

        sprintf(absPath, "%s/%s", dirPath, d->d_name);    /* 절대 경로를 저장한다. */
        if (stat(absPath,&buf) == -1)
        {
            perror("find: ");
        }
        /* 디렉토리라면 디렉토리 검색 함수를 재귀적으로 호출한다. */
        if(S_ISDIR(buf.st_mode))
        {
            FindEntry(absPath,findName);
        }

        /* 'name' 옵션의 경우 */
        if( (optionFlag == OPTION_NAME) && (strcmp(d->d_name,findName) == 0) )
        {
            printf("%s\n",absPath);
        }

        /* 'user' 옵션의 경우 */
        else if( optionFlag == OPTION_USER )
        {
            uName = getpwuid(buf.st_uid);    /* 해당하는 uid값의 user name을
                                              password file에서 가져온다. */
            if( strcmp(uName->pw_name,findName) == 0)
            {
                printf("%s\n",absPath);
            }
        }
        /* 'type' 옵션의 경우 */
        else if( optionFlag == OPTION_TYPE )
        {
            /* 타입 검사를 진행한다. */
            switch (buf.st_mode & S_IFMT)
            {
                // Directory
                case S_IFDIR:
                    mode=DI;
                    break;
            }
        }
    }
}

```

```

        // Block Device
        case S_IFBLK:
            mode=BL;
            break;
        // Char Device
        case S_IFCHR:
            mode=CH;
            break;
        // Symbol
        case S_IFLNK:
            mode=LN;
            break;
        // Regular file
        default:
            mode=RE;
    }

    /* 찾고자하는 타입을 찾았다면 */
    if( mode==findName[0] )
    {
        printf("%s\n",absPath);
    }
}

closedir(dir);
return 0;
}

/*
 *   find 명령어
 */
int main(int argc, char **argv)
{
    char *optionTmp = argv[1]; /* Temporally get option */
    char *name;
    char *curD = getcwd(NULL, BUFSIZ); /* Current Directory */

    /* Option Check */
    if(argc > 1)
    {
        /* Existing Option */
        if(optionTmp[0] == '-')
        {
            name = argv[2];
            /* 'name' Option */
            if(strcmp(optionTmp,"-name") == 0 ) optionFlag=0;
            /* 'user' Option */

```

```

        else if(strcmp(optionTmp,"-user") == 0 ) optionFlag=1;
        /* 'type' Option */
        else if(strcmp(optionTmp,"-type") == 0 ) optionFlag=2;
    }
    /* Not Existing Option */
    else
    {
        name = argv[1];
    }
}
else
{
    printf("find: Wrong Command\n");
    exit(1);
}

FindEntry(curD, name); /* 입력된 디렉토리로부터 찾고자하는 파일및 디렉토리를 찾는함수 */
return 0;
}

```