

결과 보고서

임베디드응용및실습.
5주차 실습 과제

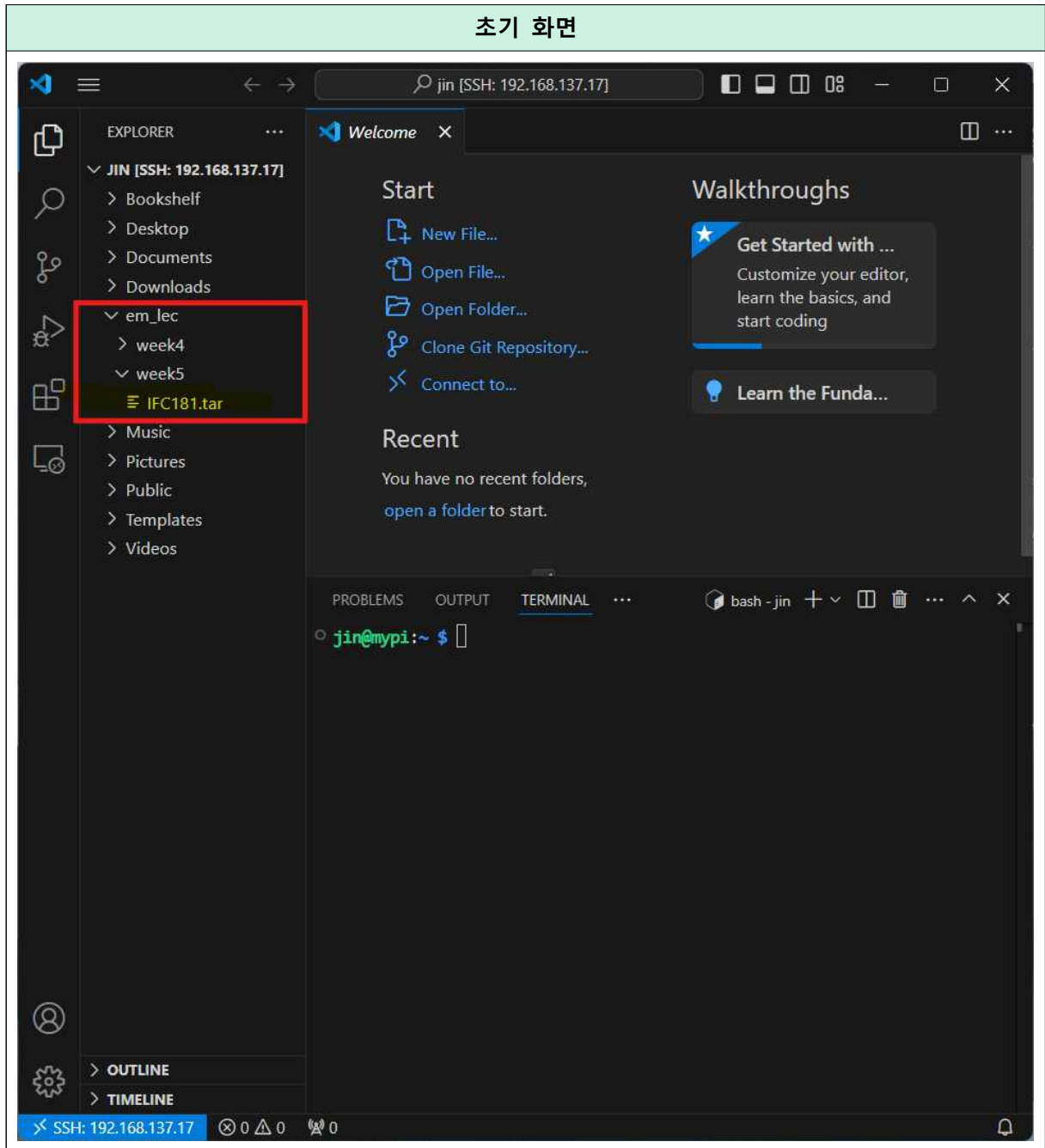
과 목 명	임베디드응용및실습
학 번	2019161068
이 름	백승진
제 출 일	2024년 10월 2일

1. IFC181.tar 파일을 설치하여, ~/embedded/week5로 옮기시오.

1) 코드

명령 작업 없음

2) 결과



3) 해석

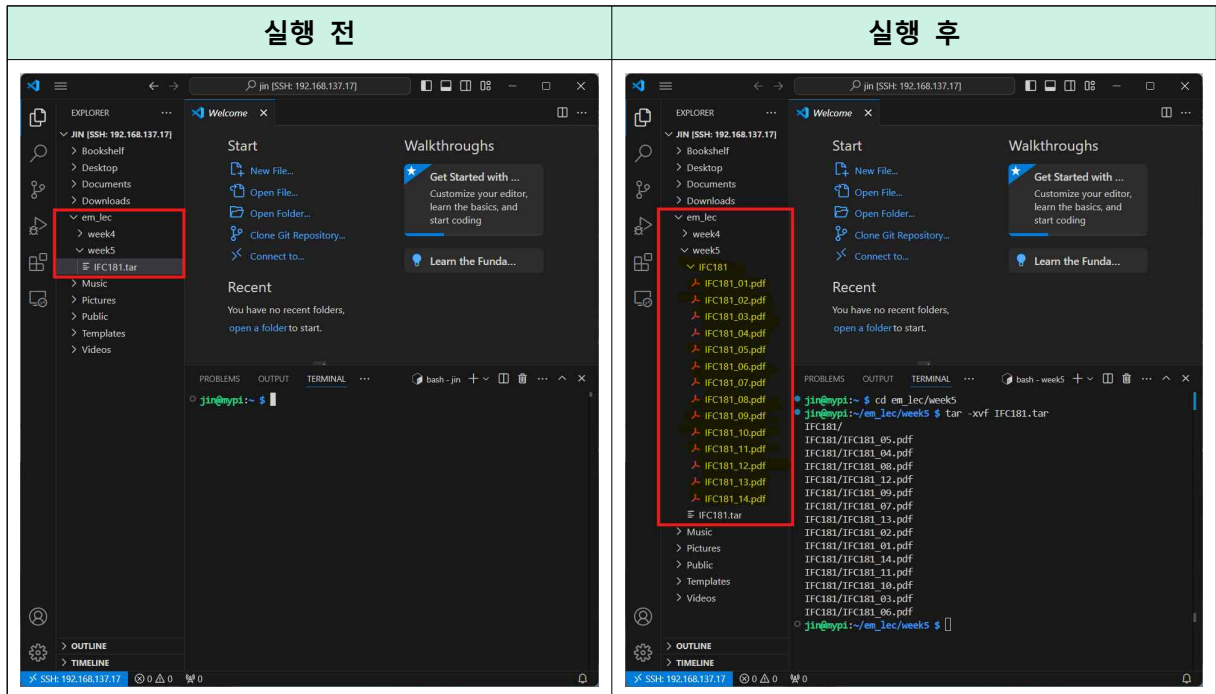
해석 없음

2. tar 명령으로 압축을 해제하시오.

1) 코드

```
cd em_lec/week5
tar -xvf IFC181.tar
```

2) 결과



3) 해석

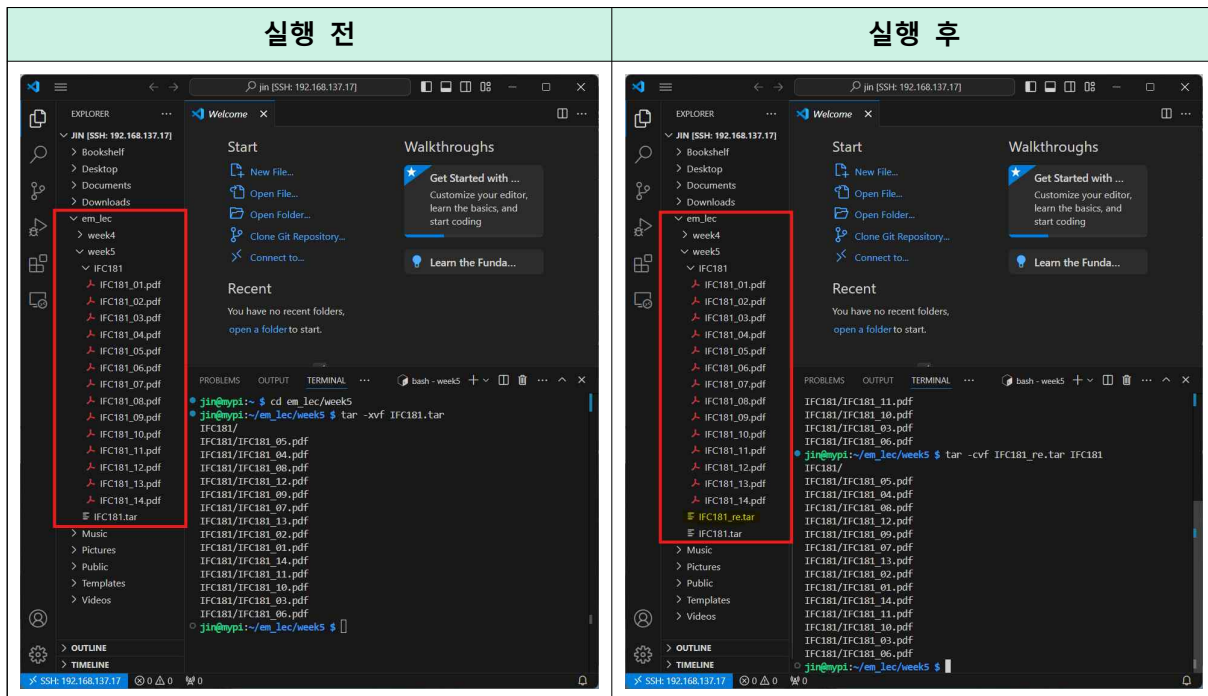
IFC181.tar 파일의 압축을 풀기 위해서 cd 명령어를 사용하여 해당 파일이 위치한 em_lec/week5 디렉토리로 이동한다. 다음으로 tar 명령어를 이용하여 IFC181.tar 파일을 압축 해제한다. 이때, 압축을 풀기 위한 옵션은 -xvf를 사용한다.

3. 문제 2에서 압축 해제된 파일들을 IFC181_re.tar로 압축하시오.

1) 코드

```
tar -cvf IFC181_re.tar IFC181
```

2) 결과



3) 해석

이전 결과에서 자동으로 IFC181 디렉터리가 생성되어, 그 안에 파일들이 있으므로 별도로 파일을 생성해서 이동시키는 과정은 생략한다.

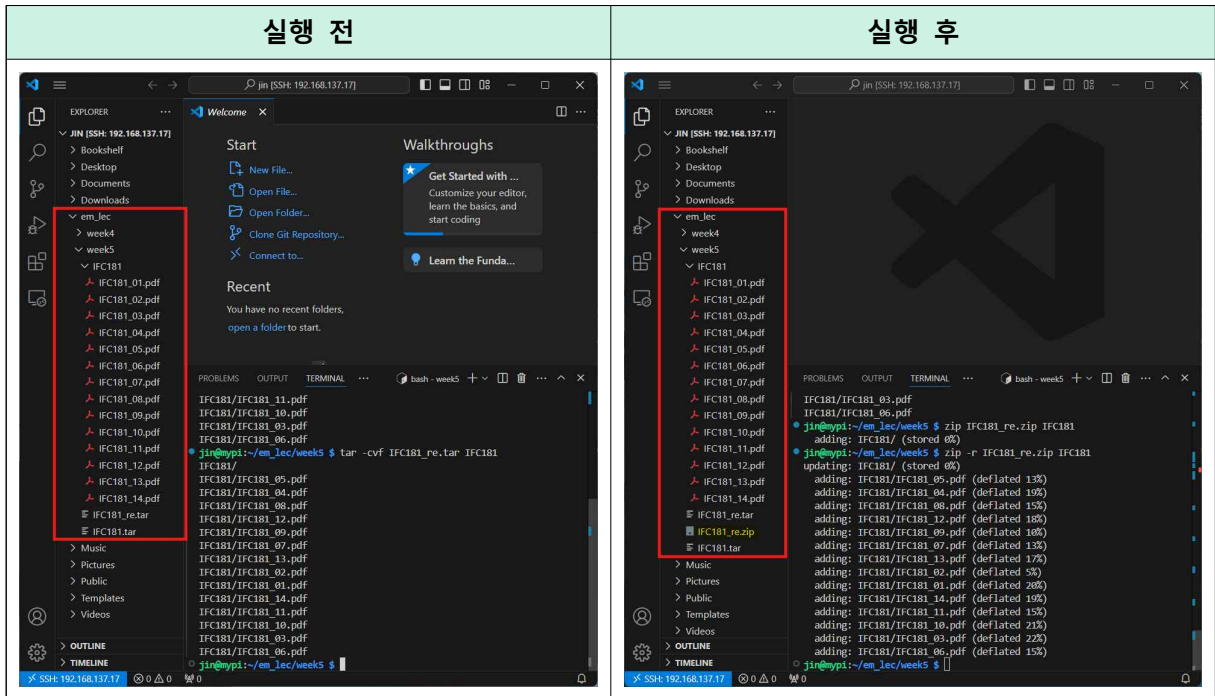
전과 동일하게 tar 명령어를 이용하여 IFC181_re.tar 파일로 압축한다. 이때, 압축을 하기 위한 옵션은 -cvf를 사용한다.

4. 문제 2에서 압축 해제된 파일들을 IFC181_re.zip으로 압축하시오.

1) 코드

```
zip -r IFC181_re.zip IFC181
```

2) 결과



3) 해석

전과는 다르게 zip 명령어를 이용하여 IFC181_re.zip 파일로 압축한다. 이때 주의할 점은 -r을 통해 지정한 디렉터리 내의 파일까지 모두 압축해야 한다는 점이다. 처음에 -r 옵션 없이 바로 압축을 시도하였는데, stored 0% 문구가 출력되어 어떤 파일도 저장되지 않았음을 확인했다. 이는 압축 과정에서 그 어떤 파일도 찾지 못했다는 뜻이므로, zip 명령어는 tar 명령어와 다르게 접근해야 함을 깨달았다.

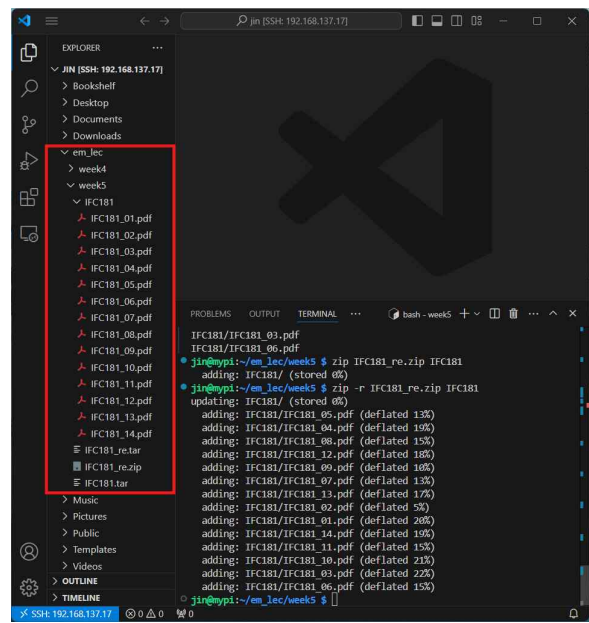
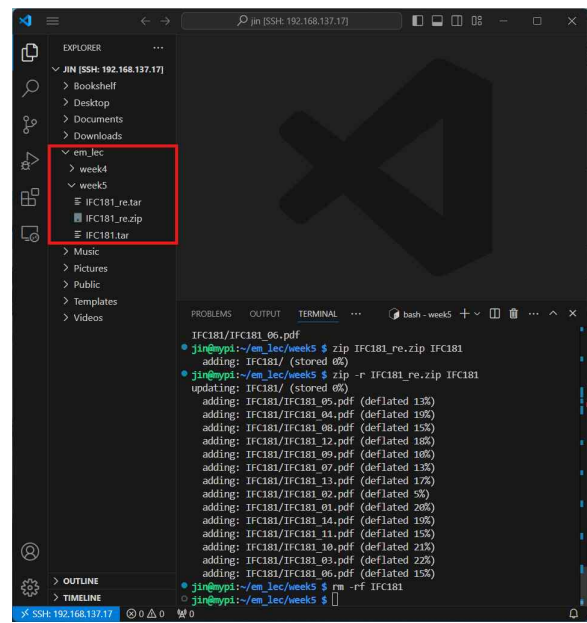
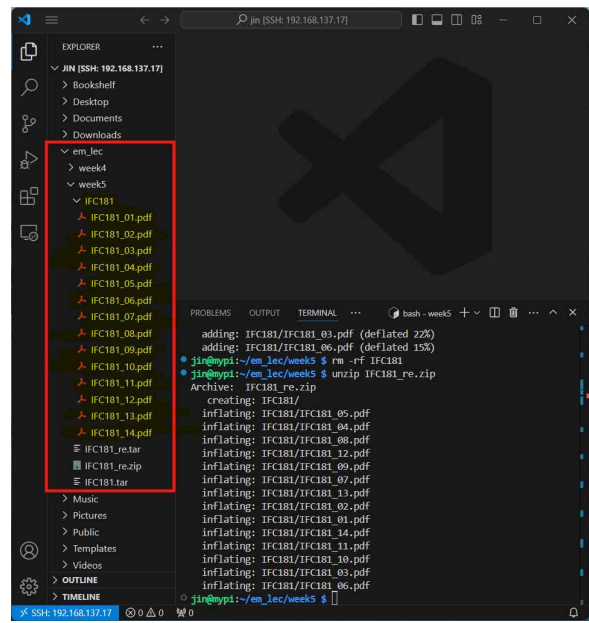
우선 zip IFC181_re.zip IFC181 명령어에서 IFC181을 압축하라고 하였음에도 IFC181을 찾지 못했으므로, IFC181 자리에는 압축할 디렉터리가 아니라 압축할 파일을 직접 입력해야 함을 알 수 있다. 이때, 직접 cd 명령어를 이용해서 IFC181 디렉터리로 이동해 파일들을 직접 압축할 수도 있지만, 다른 방법도 있다. 바로, rm 명령어로 디렉터를 삭제할 때처럼 -r 옵션을 붙여주는 것이다. 이 과제에서는 후자의 방법을 사용한다.

5. 문제 4에서 얻은 IFC181_re.zip 파일을 unzip 명령을 통해 압축 해제하시오.

1) 코드

```
rm -rf IFC181
unzip IFC181_re.zip
```

2) 결과

실행 전	삭제 후
	
압축 해제 후	
	

3) 해석

먼저 unzip을 하게 되면 기존에 있던 IFC181 디렉터리와 디렉터리명이 겹치므로, 기존의 IFC181 디렉터리를 삭제한 후에 unzip 명령어를 이용하여 압축을 해제한다.

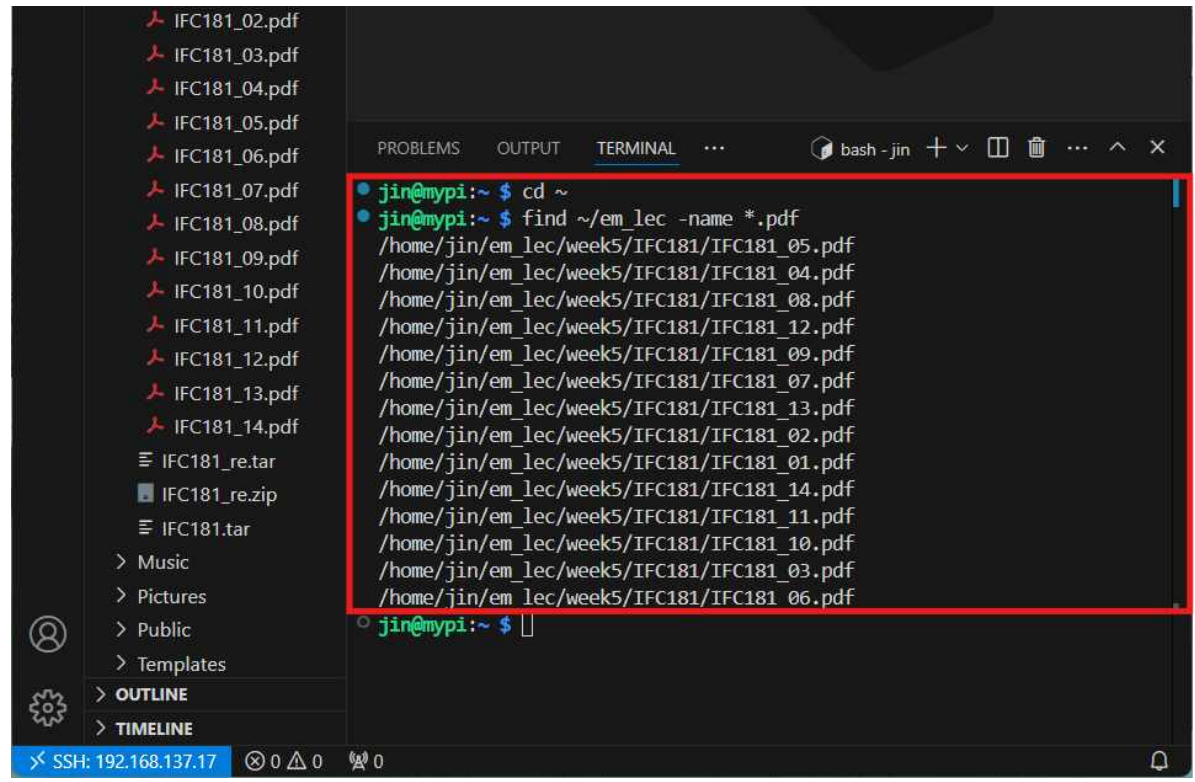
6. ~ 디렉터리로 이동한 후 find 명령을 통해 embedded 폴더에서 .pdf 파일을 찾는 명령을 수행하시오.

1) 코드

```
cd ~  
find em_lec/ *.pdf
```

2) 결과

출력 화면



```
jin@myipi:~ $ cd ~  
jin@myipi:~ $ find ~/em_lec -name *.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_05.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_04.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_08.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_12.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_09.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_07.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_13.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_02.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_01.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_14.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_11.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_10.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_03.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_06.pdf  
jin@myipi:~ $
```

3) 해석

문제에서 제시한 순서대로, cd 명령어를 이용하여 홈 디렉터리로 이동한 후, find 명령어를 이용하여 em_lec 디렉터리에 있는 pdf 파일들을 찾는다. 이때, 확장자명을 이용하므로 옵션은 -name을 사용한다.

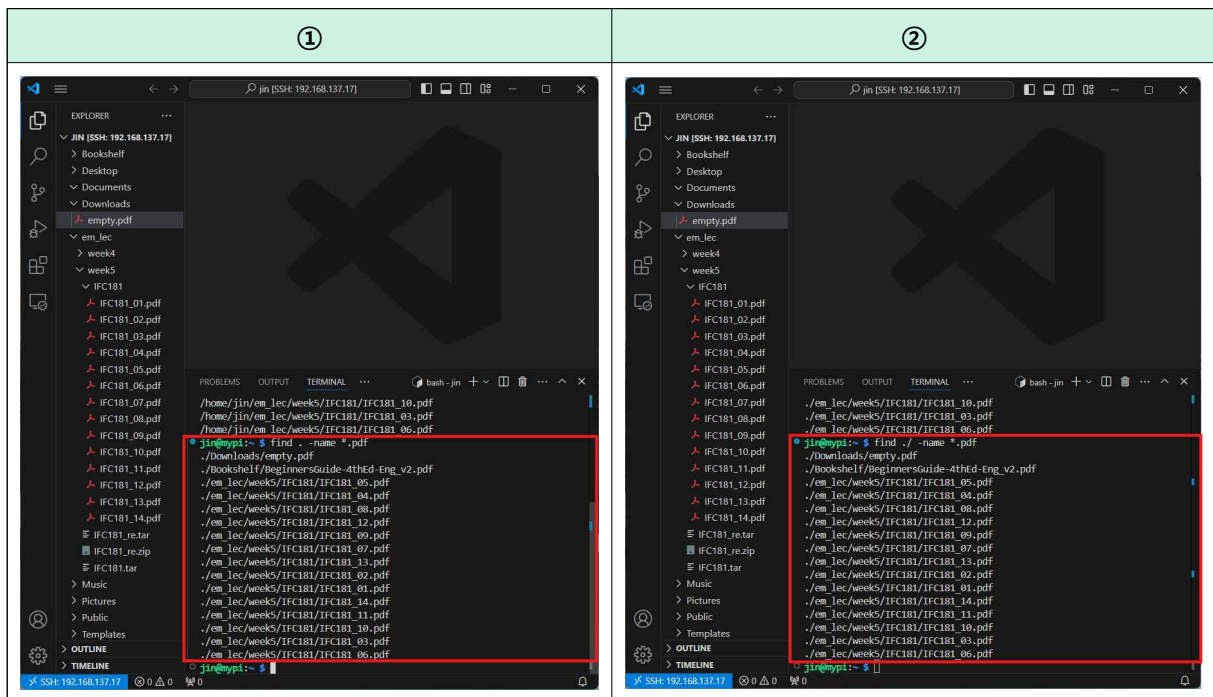
7. 아래의 명령 결과가 동일한지 아닌지 결과를 보이고 동작 결과를 설명하시오.

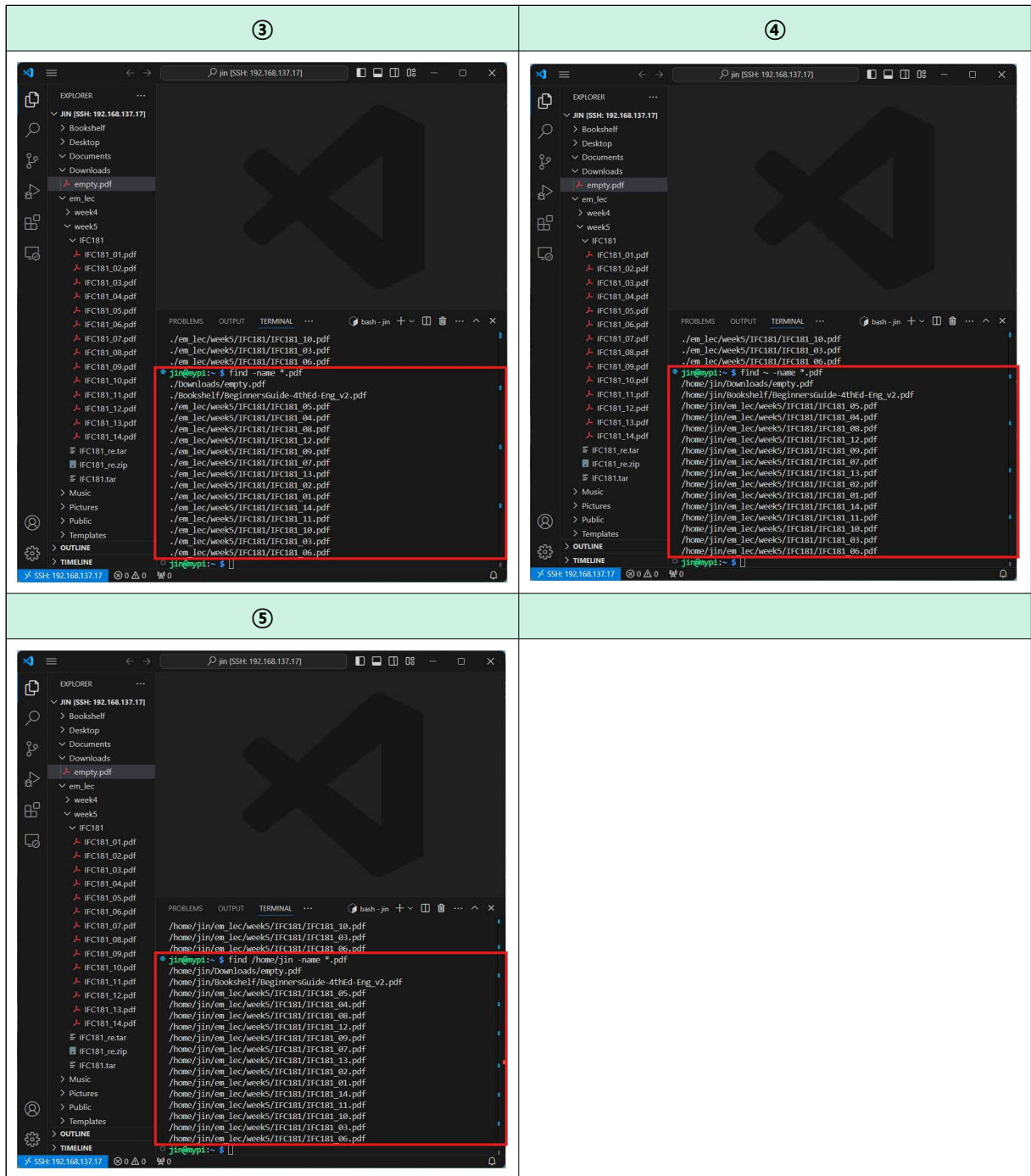
- ① innosm@innosm:~ \$find . -name *.pdf
- ② innosm@innosm:~ \$find ./ -name *.pdf
- ③ innosm@innosm:~ \$find -name *.pdf
- ④ innosm@innosm:~ \$find ~ -name *.pdf
- ⑤ innosm@innosm:~ \$find /home/innosm -name *.pdf

1) 코드

- ① find . -name *.pdf
- ② find ./ -name *.pdf
- ③ find -name *.pdf
- ④ find ~ -name *.pdf
- ⑤ find /home/jin -name *.pdf

2) 결과





3) 해석

실행 결과는 ①, ②, ③의 출력이 같고, ④, ⑤ 출력이 같다. 두 결과의 차이점은 현재 위치를 기준으로 파일의 위치를 알려주는지, home 디렉터리를 기준으로 파일의 위치를 알려주는지 정도 외에는 없다. 명령어도 마찬가지로 파일의 경로 외에는 모두 동일한 조건에서 실행하였다.

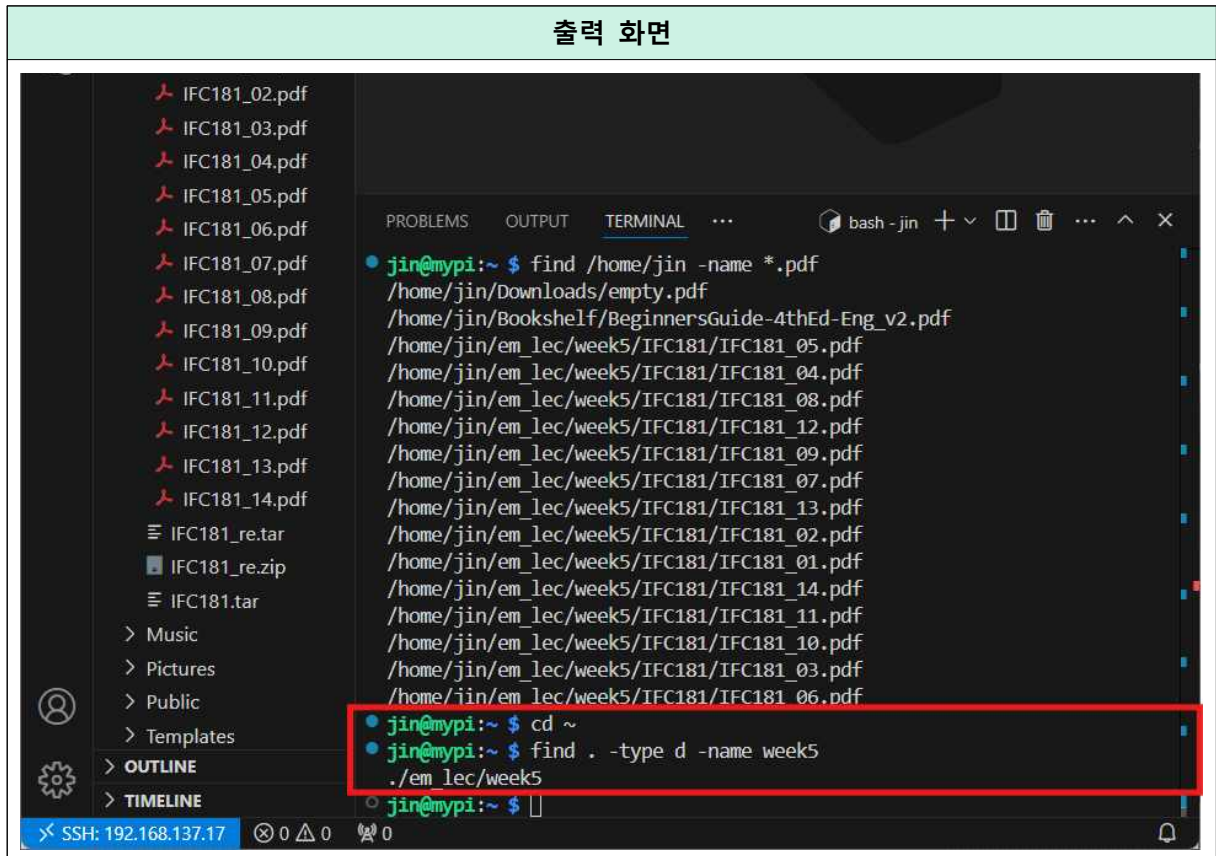
①의 '.' 명령어는 현재 디렉터리를 의미하며, ②의 '/' 명령어는 현재 디렉터리에 있는 디렉터리나 파일들을 의미한다. 결과적으로 둘은 모두 현재 디렉터리를 기준으로 명령을 실행하므로 같은 결과를 출력한다. 다음으로 ③과 같이 아무 경로도 지정하지 않은 경우에도 현재 디렉터리를 기준으로 명령을 실행한다. ④의 '~' 명령어는 홈 디렉터리를 가리키므로 홈 디렉터리를 기준으로 명령을 실행하며, ⑤의 '/home/jin' 명령어는 직접 절대 경로를 지정했으므로 해당 디렉터리를 기준으로 명령을 실행한다. 이때 /home/jin은 홈 디렉터리와 동일한 경로이므로, 우연히 ④와 같은 결과를 출력했을 뿐이다.

8. ~ 디렉터리로 이동한 후 적절한 명령을 수행하여 week5 폴더가 있는지 검색하고 결과를 보
이시오.

1) 코드

```
cd ~  
find . -type d -name week5
```

2) 결과



3) 해석

원하는 디렉터리나 파일을 찾기 위해서는 find 명령어를 이용한다. 이때, week5 디렉터리를 찾고자 하므로 -type 옵션을 d(directory)로 설정한다.

9. 아래 명령을 수행한 결과를 보이시오.

```
df -h
```

1) 코드

```
df -h
```

2) 결과

출력 화면

```
jin@mypi:~$ cd ~
jin@mypi:~$ find . -type d -name week5
./em lec/week5
jin@mypi:~$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/root	15G	3.7G	10G	27%	/
devtmpfs	1.8G	0	1.8G	0%	/dev
tmpfs	1.9G	0	1.9G	0%	/dev/shm
tmpfs	1.9G	8.6M	1.9G	1%	/run
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	1.9G	0	1.9G	0%	/sys/fs/cgroup
/dev/mmcblk0p1	253M	49M	204M	20%	/boot
tmpfs	384M	8.0K	384M	1%	/run/user/1000

3) 해석

df는 디렉터리가 있는 파일시스템의 디스크 사용량 정보를 출력하는 명령어이다. 이때 위치를 설정하지 않았으므로 현재 위치 즉, 홈 디렉터리다. Filesystem에서는 파일 시스템들의 이름들을 확인할 수 있고, Size에서는 각 파일시스템의 총용량을 확인할 수 있다. 다음으로 Used는 현재 사용 중인 공간의 크기이며, Available은 사용할 수 있는 공간의 크기로, 총용량과 사용 중인 용량의 차로 계산할 수 있다. 마지막으로 Mounted on은 현재 마운트 된 경로이다.

10. 아래 명령을 수행한 결과를 보이시오.

```
cd ~  
cd embedded  
du -h
```

1) 코드

```
cd ~/em_lec  
du -h
```

2) 결과

출력 화면

The screenshot shows a terminal window with a file manager on the left and a terminal on the right. The file manager lists files like IFC181_02.pdf through IFC181_14.pdf, IFC181_re.tar, IFC181_re.zip, and IFC181.tar. The terminal shows the following commands and output:

```
jin@mypi:~$ find /home/jin -name *.pdf  
/home/jin/Downloads/empty.pdf  
/home/jin/Bookshelf/BeginnersGuide-4thEd-Eng_v2.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_05.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_04.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_08.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_12.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_09.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_07.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_13.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_02.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_01.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_14.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_11.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_10.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_03.pdf  
/home/jin/em_lec/week5/IFC181/IFC181_06.pdf  
jin@mypi:~$ cd ~  
jin@mypi:~$ find . -type d -name week5  
./em_lec/week5  
jin@mypi:~$ cd ~/em_lec  
jin@mypi:~/em_lec$ du -h  
12K    ./week4  
4.0M   ./week5/IFC181  
16M    ./week5  
16M    .  
jin@mypi:~/em_lec$
```

3) 해석

cd 명령어를 이용하여 em_lec 디렉터리로 이동 후, du 명령어와 -h 옵션을 이용하여 사용량을 보기 좋게 출력한다. 출력 결과, em_lec 디렉터리의 하위 디렉터리인 week4와 week5의 사용량이 각각 12K, 16M으로 나타났고, week5의 하위 디렉터리는 4M로 나타났다. 마지막으로 em_lec 디렉터리의 사용량은 16M로 나타났다.

11. 문제 10에서 현재 embedded 디렉터리의 총사용량은? (하위 디렉터리 포함, 단위 표시)

1) 코드

코드 없음

2) 결과

계산 과정 및 결과

$$\begin{aligned} 12K + 4.0M + 16M + 16M &= 12K + (4 \times 1024)K + (16 \times 1024)K + (16 \times 1024)K \\ &= 12K + 4096K + 16384K + 16384K \\ &= 36876K \end{aligned}$$

따라서, em_lec 디렉터리의 총사용량은 36876 Kb다.

3) 해석

해석 없음

12. cd embedded/week5를 수행하여 위치를 이동하고, 아래 명령을 차례로 수행하시오.

- ① df .
- ② 이 디렉터리에 temp_file 파일을 생성하고 파일에 1을 기록
- ③ stat temp_file 명령을 통해 파일의 크기를 확인
- ④ df .

1) df .

① 코드

```
cd ~/em Lec/week5
df .
```

② 결과

출력 화면

```
jin@mypi:~$ cd ~
jin@mypi:~$ find . -type d -name week5
./em Lec/week5
jin@mypi:~$ cd ~/em Lec
jin@mypi:~/em Lec$ du -h
12K  ./week4
4.0M  ./week5/IFC181
16M  ./week5
16M  .
jin@mypi:~/em Lec$ cd ~/em Lec/week5
jin@mypi:~/em Lec/week5$ df .
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/root        14986204 3840676  10484020  27% /
```

③ 해석

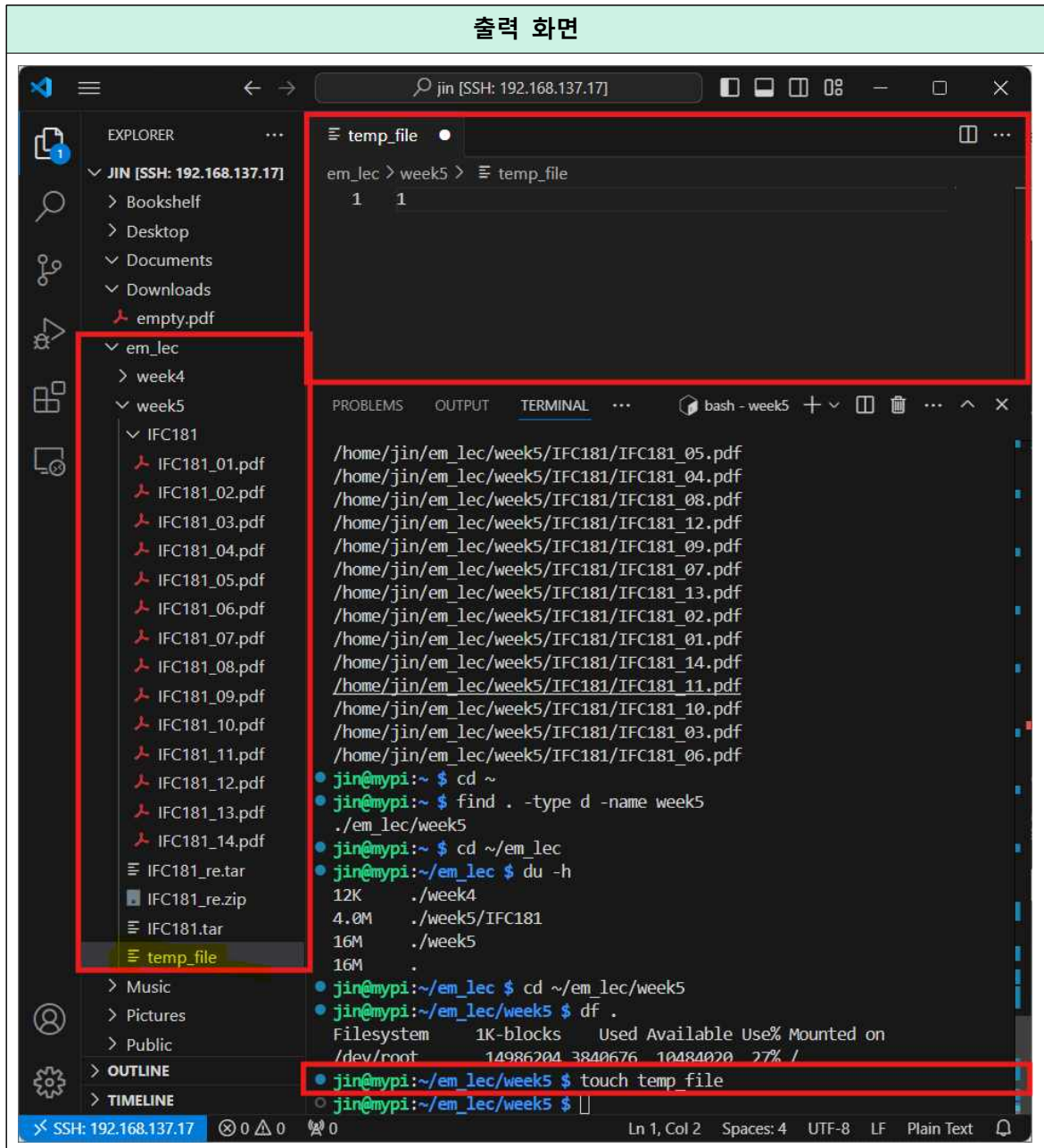
df는 디렉터리가 있는 파일시스템의 디스크 사용량 정보를 출력하는 명령어이다. 이때 위치는 . 명령어를 이용하였으므로 현재 위치 즉, week5 디렉터리이다. 내용을 보면, Filesystem에 /dev/root가 있는데, 이는 week5 디렉터리가 위치한 디스크 장치의 이름이다. 다음으로 1K-blocks는 week5 디렉터리가 위치한 파일시스템의 총용량으로, KB 단위로 나타난다. 즉, 14986204는 약 14.6GB이다. 다음으로 Used는 사용 중인 용량이며, Available은 사용할 수 있는 공간으로, 총용량과 사용 중인 용량의 차로 계산할 수 있다. 마지막으로 Mounted on은 현재 마운트 된 경로로, '/'는 루트 디렉터리 즉 최상위 디렉터리에 마운트 되어 있음을 나타낸다.

2) 이 디렉터리에 temp_file 파일을 생성하고 파일에 1을 기록

① 코드

```
touch temp_file
```

② 결과



③ 해석

현재 위치한 week5 디렉터리에 touch 명령어를 이용하여 temp_file 파일을 생성하고, 해당 파일에 직접 1을 입력하여 저장한다.

3) stat temp_file 명령을 통해 파일의 크기를 확인

① 코드

```
stat temp_file
```


② 결과

출력 화면

```

/home/jin/em Lec/week5/IFC181/IFC181_11.pdf
/home/jin/em Lec/week5/IFC181/IFC181_10.pdf
/home/jin/em Lec/week5/IFC181/IFC181_03.pdf
/home/jin/em Lec/week5/IFC181/IFC181_06.pdf
jin@mypi:~$ cd ~
jin@mypi:~$ find . -type d -name week5
./em Lec/week5
jin@mypi:~$ cd ~/em Lec
jin@mypi:~/em Lec$ du -h
12K  ./week4
4.0M  ./week5/IFC181
16M   ./week5
16M   .
jin@mypi:~/em Lec$ cd ~/em Lec/week5
jin@mypi:~/em Lec/week5$ df .
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/root        14986204 3840676 10484020 27% /
jin@mypi:~/em Lec/week5$ touch temp_file
jin@mypi:~/em Lec/week5$ stat temp_file
File: temp_file
Size: 1          Blocks: 8          IO Block: 4096   일반 파일
Device: b302h/45826d Inode: 396681     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   jin)   Gid: ( 1000/   jin)
Access: 2024-10-02 18:13:41.959752350 +0900
Modify: 2024-10-02 18:13:49.809673409 +0900
Change: 2024-10-02 18:13:49.809673409 +0900
Birth: -
jin@mypi:~/em Lec/week5$
  
```

③ 해석

stat 명령어를 이용하여 temp_file의 속성을 확인한다. Size는 파일의 크기를 나타내며, 단위는 Byte이다. 즉 파일의 크기는 1 Byte라고 할 수 있다.

4) df .

① 코드

df .

② 결과

초기 df . 출력 화면	파일 추가 후 df . 출력 화면
<pre> jin@mypi:~/em Lec/week5\$ df . Filesystem 1K-blocks Used Available Use% Mounted on /dev/root 14986204 3840676 10484020 27% / </pre>	<pre> jin@mypi:~/em Lec/week5\$ df . Filesystem 1K-blocks Used Available Use% Mounted on /dev/root 14986204 3840676 10484004 27% / </pre>

③ 해석

처음 df .의 출력 값과 temp_file 파일을 생성한 후 df .의 출력 값에 차이가 있다. 바로 Used와 Available 값이다. 앞에서 설명했듯, Used는 사용 중인 공간이며, Available은 사용할 수 있는 공간이다. 따라서 처음 Used의 출력 값보다 파일을 생성한 후의 Used 출력 값은 더 늘어날 것이고, Available 값은 반대로 처음 출력 값보다 더 줄어든 것이다. 또한, 당연히 Used의 변화량과 Available의 변화량은 같으며, **그 값은 16KB로 확인할 수 있다.**

여기서 16KB는 예상하지 못한 값이다. temp_file 파일의 크기는 1바이트이기 때문에 최소 블록 단위인 4KB만큼 할당되었을 것이다. 또한, 해당 파일은 디스크에 8개의 블록 즉, 32KB만큼 할당되었다. 따라서 16KB는 어디서도 나올 수 없는 숫자이다. 이에 대해 원인을 찾아보았으나, 인터넷에서는 결국 찾을 수 없었고, GPT에서도 명확한 답을 얻을 수 없었다.

5) 오류 검출 (실패)

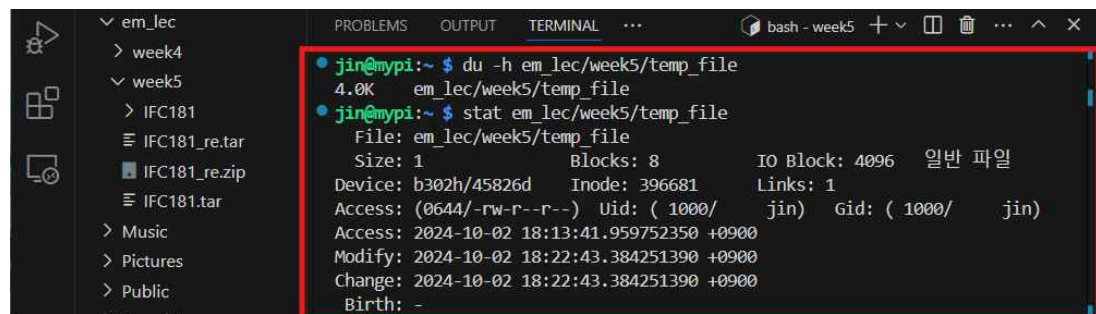
① temp_file의 크기와 디스크 사용량 재확인

i) 코드

```
cd ~  
du -h em_lec/week5/temp_file  
stat em_lec/week5/temp_file
```

ii) 결과

결과 화면



```
jin@mypi:~$ du -h em_lec/week5/temp_file  
4.0K  em_lec/week5/temp_file  
jin@mypi:~$ stat em_lec/week5/temp_file  
File: em_lec/week5/temp_file  
Size: 1          Blocks: 8          IO Block: 4096   일반 파일  
Device: b302h/45826d  Inode: 396681      Links: 1  
Access: (0644/-rw-r--r--)  Uid: ( 1000/   jin)   Gid: ( 1000/   jin)  
Access: 2024-10-02 18:13:41.959752350 +0900  
Modify: 2024-10-02 18:22:43.384251390 +0900  
Change: 2024-10-02 18:22:43.384251390 +0900  
Birth: -
```

iii) 해석

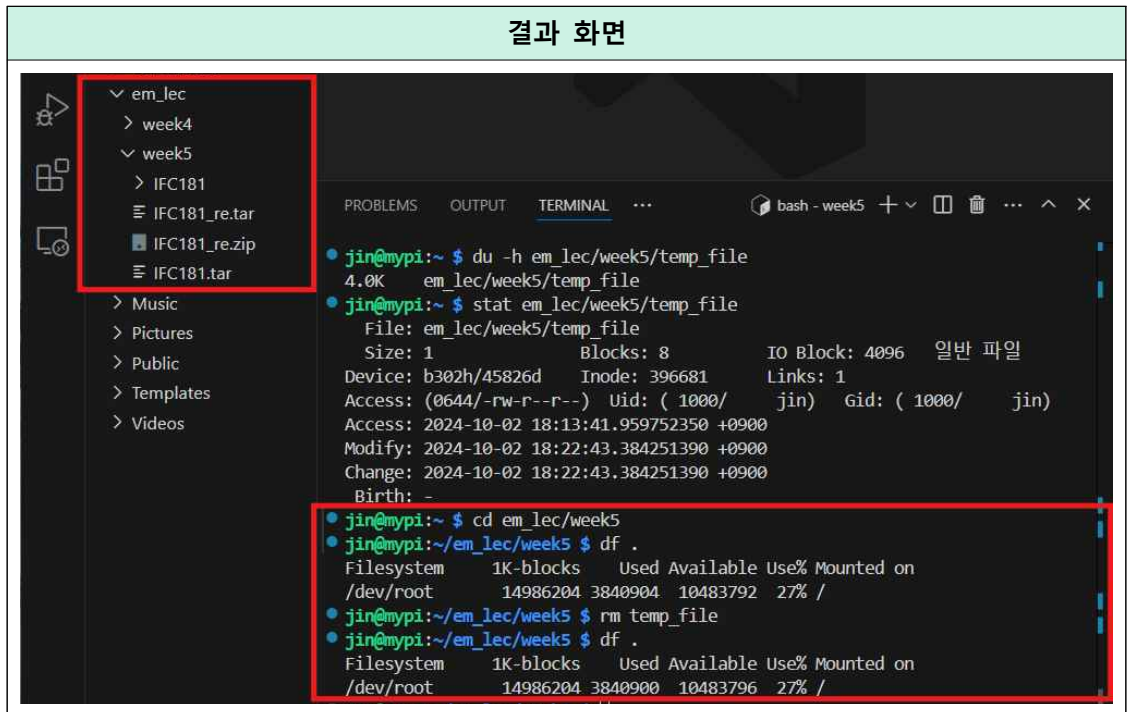
우선 du와 stat 명령어를 이용하여 temp_file 파일의 크기와 디스크 사용량을 확인하였다. 파일 크기는 1Byte, 사용량은 4KB로 결과는 이론값과 같았다. 즉, 파일에 문제는 없었다.

② 현재 week5의 디스크 공간 정보와 temp_file 파일 삭제 후 week5의 디스크 공간 정보 비교

i) 코드

```
df .  
rm temp_file  
df .
```

ii) 결과



iii) 해석

df 명령어를 통해 확인한 결과, temp_file 파일을 삭제하기 전의 Used 값은 3840904이며 Available 값은 10483792이다. 그리고, rm 명령어를 통해 temp_file 파일을 삭제한 후, 다시 확인한 결과, temp_file 파일을 삭제한 후의 Used 값은 3840900이며 Available 값은 10483796이다. 즉, 이론값대로, Used와 Available 모두 4KB의 변화가 있었다. 즉, 파일의 크기는 4KB가 맞음을 재확인할 수 있었다.

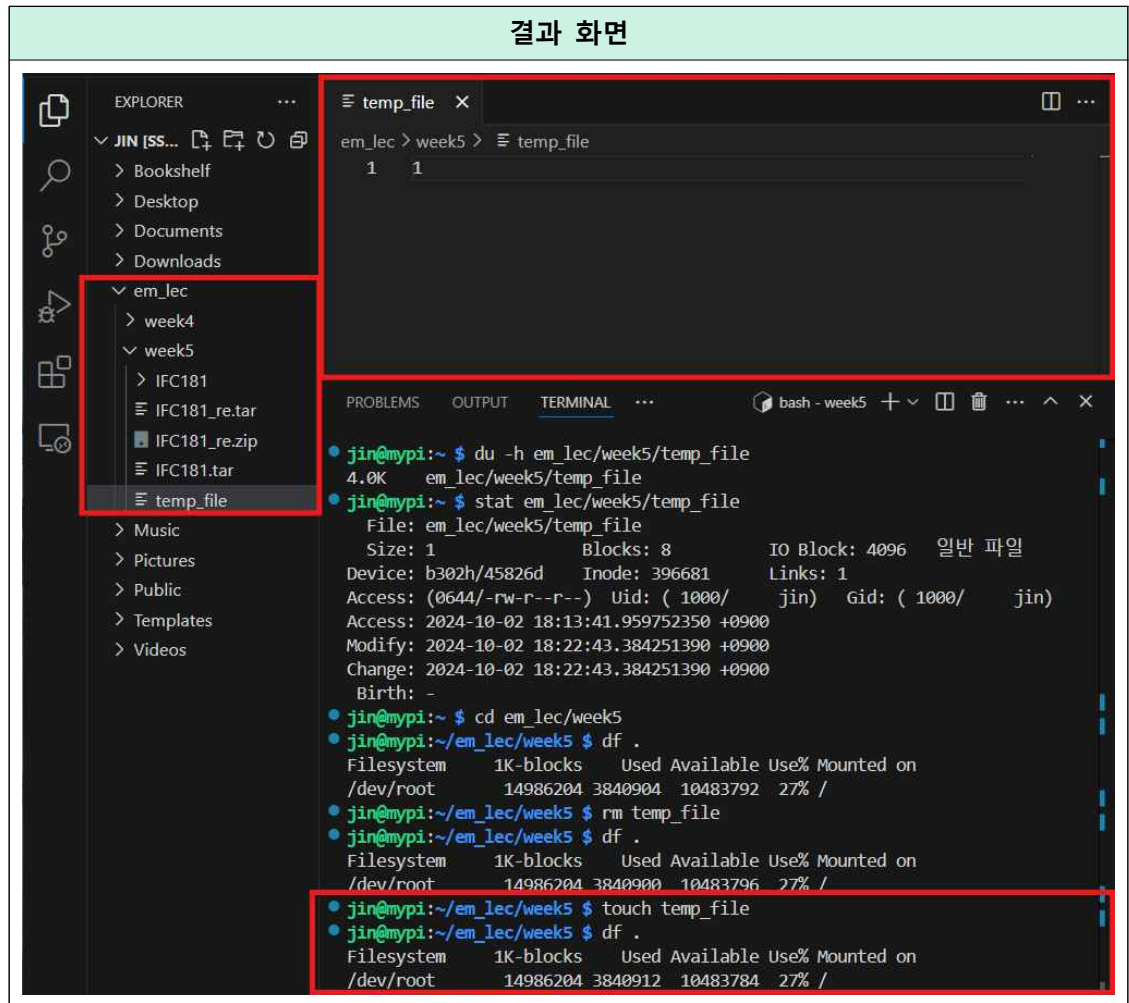
하지만 이 과정만으로는 왜 이전에는 16KB의 변화가 있었는지를 설명할 수 없다. 또한, 한 가지 특이한 점은 VSCode를 종료 직전의 Used 값과, 재접속했을 때의 Used 값이 다르다는 점이다. 첫 시도에서 최종 Used 값은 3840692인데, 이번 시도에서 첫 Used 값은 3840904로 완전히 달랐다. 즉, 이 공간은 유동적으로 변경될 수 있다는 점을 생각해둘 필요가 있다.

③ 현재 week5의 디스크 공간 정보와 temp_file 파일 재생성 후 week5의 디스크 공간 정보

i) 코드

```
touch temp_file
df .
```

ii) 결과



iii) 해석

temp_file을 재생성 후 다시 df 명령어로 확인했을 때는 Used와 Available 값에 12KB의 변화가 있었다. 즉, 첫 번째 시도했을 때와도 다른 결과가 나와버렸다. 왜 이러한 결과가 나온 것일까? 여기에는 몇 가지 가설을 설정할 수 있다. 우선 첫 번째 가설은 실제 데이터를 저장하는 공간 외에 메타데이터를 저장하는 데에 추가 공간이 요구되므로 사실 더 많은 공간을 할당받는다는 것이다. 두 번째 가설은 최소 블록 단위라는 개념 외에도 2차적으로 여유 공간을 고려하여 할당받는다는 것이다. 강의 자료의 16페이지를 보면, Ext2 파일시스템은 여러 블록 그룹으로 이루어지고, 그 그룹 중의 일부가 데이터 블록이기 때문에 첫 번째 가설을 잘 설명할 수 있다. 하지만, 만약 그렇다면, 파일을 삭제했을 때 디렉터리 공간에 4KB의 변화가 있는 것은 설명할 수 없다. 물론 파일을 삭제할 때 더미 데이터가 남아 12KB 중 4KB만 삭제된다면 모든 게 맞아떨어지지만, 이를 증명할 방법은 찾지 못했다. 마찬가지로 두 번째 가설의 경우에도 첫 번째 가설과 마찬가지로 삭제할 때 회수되는 공간이 4KB인 것은 설명하지 못한다. 원인을 파악하기 위해서는 조금 더 생각해볼 필요가 있을 듯하다.

13. 현재 사용하는 라즈비언 OS가 몇 비트 시스템인지 확인하고 결과를 첨부하시오.

1) 코드

```
cd ~  
uname -m  
getconf LONG_BIT
```

2) 결과

출력 결과

```
jin@mypi:~/em_lec $ cd ~  
jin@mypi:~/em_lec $ df .  
Filesystem      1K-blocks    Used Available Use% Mounted on  
/dev/root        14986204 3840676 10484020 27% /  
jin@mypi:~/em_lec/week5 $ touch temp_file  
jin@mypi:~/em_lec/week5 $ stat temp_file  
File: temp_file  
Size: 1          Blocks: 8          IO Block: 4096   일반 파일  
Device: b302h/45826d Inode: 396681     Links: 1  
Access: (0644/-rw-r--r--)  Uid: ( 1000/   jin)   Gid: ( 1000/   jin)  
Access: 2024-10-02 18:13:41.959752350 +0900  
Modify: 2024-10-02 18:13:49.809673409 +0900  
Change: 2024-10-02 18:13:49.809673409 +0900  
Birth: -  
jin@mypi:~/em_lec/week5 $ df .  
Filesystem      1K-blocks    Used Available Use% Mounted on  
/dev/root        14986204 3840676 10484020 27% /  
jin@mypi:~/em_lec/week5 $ cd ~  
jin@mypi:~ $ uname -m  
armv7l  
jin@mypi:~ $ getconf LONG_BIT  
32  
jin@mypi:~ $
```

3) 해석

uname -m 명령어를 입력하였으나, armv7l이 출력되었다. 이는 ARM 아키텍처의 버전이라고 하는데, v7은 7세대, v8은 8세대를 의미한다. 보통 v7은 라즈베리파이와 같은 소형 장치에서 사용되며 32비트 프로세서에서 사용된다. 반대로 v8은 스마트폰이나 태블릿, 노트북 등에서 사용되며 64비트 프로세서에서 사용된다. 즉, 현재 노트북으로 사용하고 있는 라즈비언 OS는 7세대로 32비트 운영체제이다. 이는 getconf LONG_BIT 명령어를 이용하여 확인할 수 있다.