

4주차 예비보고서

전공 : 경영학과

학년 : 4학년

학번 : 20190808

이름 : 방지혁

1. UNIX 시스템에 접속해본 뒤 자신의 힐 디렉토리를 확인해본다. UNIX 시스템에 접속해본 뒤

자신의 흄 디렉토리를 확인해본다.

```
System load: 0.14          Temperature: 36.0 C
Usage of /: 2.8% of 438.01GB Processes: 995
Memory usage: 4%           Users logged in: 1
Swap usage: 0%             IPv4 address for eno2: 172.30.10.5

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

12 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

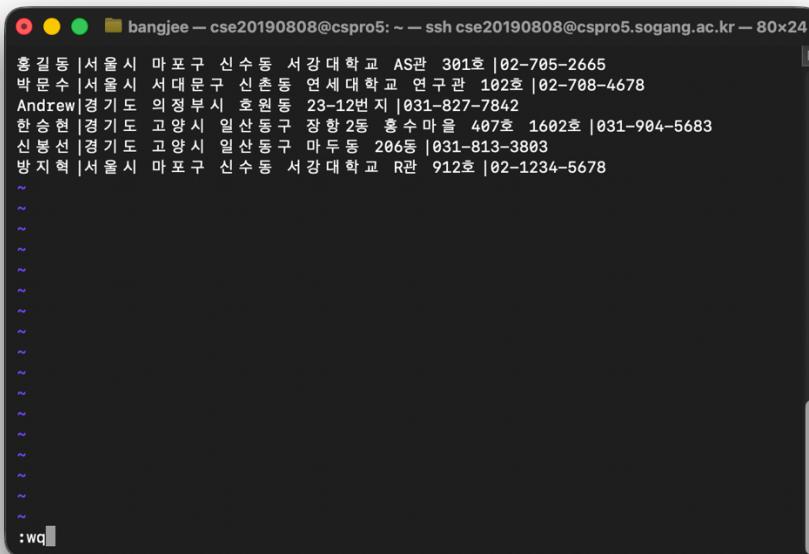
Last login: Tue Sep 16 09:47:20 2025 from 49.170.81.60
[cse20190808@cspro5:~] ssh cse20190808@cspro5.sogang....
[cse20190808@cspro5:~] echo $HOME
/sogang/under/cse20190808
[cse20190808@cspro5:~]
```

2. 쉘 프로그래밍 실험에서 사용할 데이터 파일인 전화번호부를 만들어본다. 단 데이터 파일의 형식은 실험에서 나온 예제에 따르도록 한다. 5명 이상이 들어가 있는 데이터를 만들되 vi 에디터를 이용하여 작성한다. 단 파일명은 data로 한다.

3. 위의 예제를 편집하는데 사용한 vi 명령어들을 나열하고, 해당 명령 수행하는 결과를 적어보도록 한다.

```
Last login: Tue Sep 16 09:47:20 2025 from 49.170.81.60
[cse20190808@cspro5:~$ echo $HOME
/sogang/under/cse20190808
[cse20190808@cspro5:~$ ls
pintos
[cse20190808@cspro5:~$ vi data
```

vi data를 shell에 입력합니다. data라는 파일을 만들 것이며 vi 에디터로 열겠다는 명령어입니다.



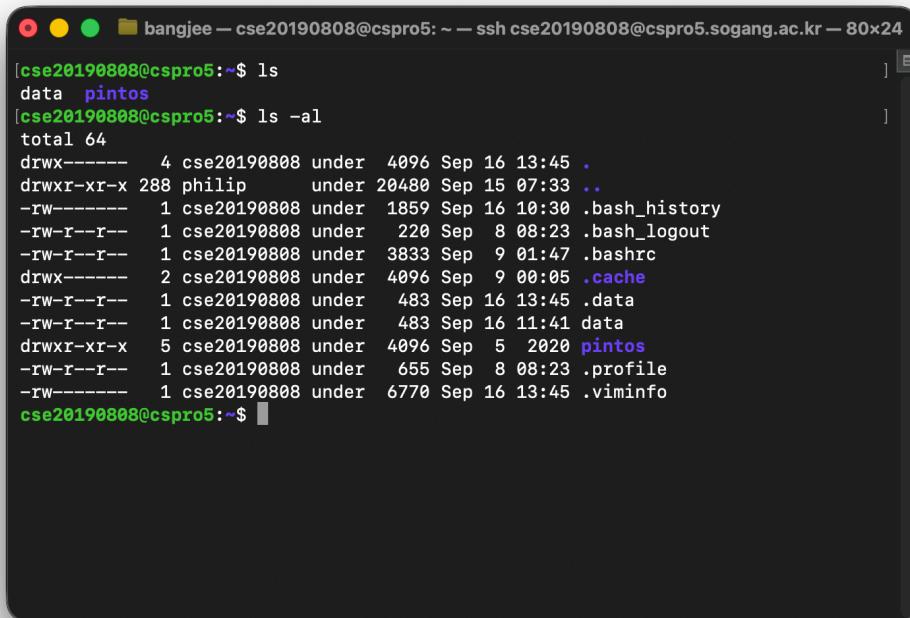
Esc 및 i를 입력하여 입력모드를 진입합니다. 이후 샘플 데이터를 입력하고 :wq를 통해 빠져나갑니다. wq는 저장하고 종료한다는 뜻입니다.

4. 위에서 작성한 데이터 파일을 \$home/.data 파일로 복사한다. 복사하기 위하여 사용한 명령들을 적어보도록 한다.

```
[cse20190808@cspro5:~$ cp data $HOME/.data
```

cp [옵션] 원본 복사본 명령어를 사용했습니다. 그렇지만 옵션을 사용할 필요가 없기에 사용하지 않았습니다.

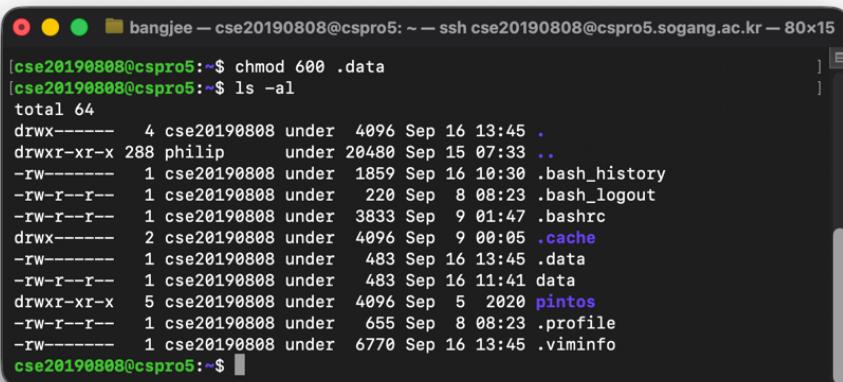
그냥 ls를 하면 안 보이지만 ls -al를 입력하면 보이는 것을 확인할 수 있습니다.



```
[cse20190808@cspro5:~]$ ls
data pintos
[cse20190808@cspro5:~]$ ls -al
total 64
drwx----- 4 cse20190808 under 4096 Sep 16 13:45 .
drwxr-xr-x 288 philip under 20480 Sep 15 07:33 ..
-rw----- 1 cse20190808 under 1859 Sep 16 10:30 .bash_history
-rw-r--r-- 1 cse20190808 under 220 Sep 8 08:23 .bash_logout
-rw-r--r-- 1 cse20190808 under 3833 Sep 9 01:47 .bashrc
drwx----- 2 cse20190808 under 4096 Sep 9 00:05 .cache
-rw-r--r-- 1 cse20190808 under 483 Sep 16 13:45 .data
-rw-r--r-- 1 cse20190808 under 483 Sep 16 11:41 data
drwxr-xr-x 5 cse20190808 under 4096 Sep 5 2020 pintos
-rw-r--r-- 1 cse20190808 under 655 Sep 8 08:23 .profile
-rw----- 1 cse20190808 under 6770 Sep 16 13:45 .viminfo
[cse20190808@cspro5:~]$
```

5. \$home/.data 파일을 그룹 및 다른 사용자가 아무 권한도 갖지 않도록 권한 변경을 해본다. 사용

용한 명령을 적어보도록 한다.



```
[cse20190808@cspro5:~]$ chmod 600 .data
[cse20190808@cspro5:~]$ ls -al
total 64
drwx----- 4 cse20190808 under 4096 Sep 16 13:45 .
drwxr-xr-x 288 philip under 20480 Sep 15 07:33 ..
-rw----- 1 cse20190808 under 1859 Sep 16 10:30 .bash_history
-rw-r--r-- 1 cse20190808 under 220 Sep 8 08:23 .bash_logout
-rw-r--r-- 1 cse20190808 under 3833 Sep 9 01:47 .bashrc
drwx----- 2 cse20190808 under 4096 Sep 9 00:05 .cache
-rw-r--r-- 1 cse20190808 under 483 Sep 16 13:45 .data
-rw-r--r-- 1 cse20190808 under 483 Sep 16 11:41 data
drwxr-xr-x 5 cse20190808 under 4096 Sep 5 2020 pintos
-rw-r--r-- 1 cse20190808 under 655 Sep 8 08:23 .profile
-rw----- 1 cse20190808 under 6770 Sep 16 13:45 .viminfo
[cse20190808@cspro5:~]$
```

chmod 600 .data 명령어를 사용했다. 4번 항목에 캡쳐한 사진과 비교하자면 기존에는 -rw-r--r--이

었지만 -rw-----로 바뀌면서 소유자 외에 다른 사람은 읽지도 쓰지도 실행하지도 못합니다.

6. 디렉토리에 대한 읽기, 쓰기, 실행 권한을 설정해보고 각각이 갖는 의미를 살펴본다.

디렉토리에 대한 읽기 권한 이라 함은 디렉토리 안의 파일 목록을 확인할 수 있는 권한으로 디렉토리가 아닌 파일에 대한 권한일 경우 파일 내용을 읽을 수 있는 권한입니다. 쓰기 권한은 디렉토리일 경우 새로운 디렉토리 또는 파일을 만들 수 있고 이름도 변경할 수 있으며 파일일 경우 파일을 수정 혹은 삭제할 수 있게 됩니다. 실행 권한의 경우 디렉토리일 경우 디렉토리 내부로 접근할 수 있는 권한으로 파일을 경우 말그대로 실행할 수 있는 권한입니다.

7. c/c++ 프로그램의 컴파일 과정에 대하여 요약하라. 각 단계별로 하는 일들과 관련된 도구들 또한 명시하라.

소스코드가 실행 가능한 프로그램이 되기 위해서는 전처리(Preprocess) → 컴파일(compile) → 어셈블(Assemble) → 링킹(Linking) 4단계를 거친다고 보면 됩니다.

첫번째 단계인 전처리 단계에 대해 설명하자면 소스 코드 파일(*.c)를 전처리된 소스 코드 파일(*.i)로 바꾸는 과정입니다. #include 지시문에 대해서는 헤더 파일을 찾아 해당되는 모든 내용들을 소스 코드에 삽입합니다. #define 지시문의 매크로에 대해 모두 실제 값으로 치환합니다. #ifdef, #ifndef에 대해서는 조건부 컴파일 처리를 하고 주석도 제거해줍니다. cpp는 대표적인 전처리기 도구이고 이는 .cpp 확장자와는 별개입니다. gcc 혹은 g++을 사용할 때 내부적으로 cpp 전처리기가 자동적으로 호출되기 때문에 gcc -E 옵션으로 전처리만 수행이 가능합니다.

두 번째 단계는 컴파일 단계입니다. *.i 파일을 입력으로 받아 이를 어셈블리어로 번역하여 *.s 파일을 생성합니다. gcc compiler가 대표적인 컴파일 도구이며 gcc -S 옵션으로 컴파일만 수행하는 것이 가능합니다.

세 번째 단계는 어셈블 단계로 *.s 파일을 입력으로 받아 *.o 파일 즉 오브젝트 파일을 생성합니다. 기존이 어셈블리 코드를 기계어로 바꿔준 것이고 as(assembler)가 유닉스 어셈블리로 이또한 gcc에 포함되어 있습니다. gcc -c 옵션으로 링킹전까지 수행 가능합니다.

마지막 단계는 링킹 단계로 .o 파일들 및 라이브러리를 받아 실행 파일(a.out 혹은 지정된 이름)을 생성합니다. 여러 오브젝트 파일을 하나로 결합하고 라이브러리와 링크 하는 것입니다. ld가 유닉스 링커로 이또한 gcc 내부에서 호출됩니다.