부록 5. Linux Command Line

Linux

💡 리눅스는 아직도 많은 사람들에게 가깝고도 먼 존재입니다. 윈도우나 맥처럼 GUI(Graphical User Interfaces) 형태의 배포판이 있기는 있지만 실제로 운용하는 리눅스는 대부분 서버 형태입니다. CLI(Command-line interfaces), 즉 명령어를 입력하는 형태로 컴퓨터를 사용해야 하는 것이죠. GUI 기반의 OS가 출시되기 이전에는 MS 社의 DOS(Disk Operating System)가 사용 되었습니다. 이 역시 CLI 기반의 운영체제였기 때문에 GUI 기반의 운영체제가 시장에 출시 되었을 때의 반응은 아주 충격적이었습니다. 아이콘을 눌러서 프로그램을 실행할 수 있다니 실로 대단한 변화가 찾아온 것이었습니다.
하지만 GUI 기반 OS를 모든 영역에서 사용하지는 않습니다. 화면으로 보여질 필요가 없는 시스템을 GUI로 구현하는 것은 리소스 낭비에 불과하기 때문입니다. 오히려 성능 저하만 일어나기 때문에 우리는 열심히 Command Line을 공부해서 OS를 운용해야 합니다.

💡 Shell 이라는 것은 쉽게 말하면 OS와 소통할 수 있는 창구입니다. 전화 수화기의 역할을 한다고 볼 수도 있겠네요. 중요한 것은 Shell을 통해 사용자의 명령이 OS에 전달되어 실행된다는 것입니다. 그리고 그 명령에는 약속된 규칙이 있습니다. Shell 을 학습한다는 것은 바로 이렇게 약속된 명령어를 작성하는 법을 익히는 것입니다. 기본적인 명령어를 본 부록을 통해 제공해 드리니 막연한 거부감과 두려움을 조금이나마 해소하셨으면 합니다.

> **출처**
> - **LinuxCommand.org** : https://linuxcommand.org/index.php
> - **Linux Shell Commands** : https://docs.cs.cf.ac.uk/notes/linux-shell-commands/

0. Prologue

0.1 Meet Linux Shell !

컴퓨터를 부팅했는데 다음과 같은 화면이 나오면 당황스러움을 감출 수 없습니다. 바탕화면도, 작업표시줄도 없는...새까만 화면이라니... 먼저 위에 표시된 부분부터 보도록 하겠습니다.

```
ggingmin@ubuntu_server:~$ [사용자명]@[컴퓨터명]:~$
```

위와 같이 명령 입력을 위해 띄워진 것을 **프롬프트(Prompt)**라고 합니다. 일반 사용자의 경우 `$` , 루트 사용자의 경우 `#` 로 표시됩니다. 자세한 내용은 뒤에서 다룰 예정이기에 지금은 가볍게 둘러본다는 생각으로 넘어가시면 됩니다.

0.2 Shell의 종류

Shell 은 사실 다양한 종류가 있습니다. 각 Shell 은 기본적으로 수행하는 역할은 거의 동일하지만 특정 작업에 있어 수행 가능여부에 차이가 있습니다.

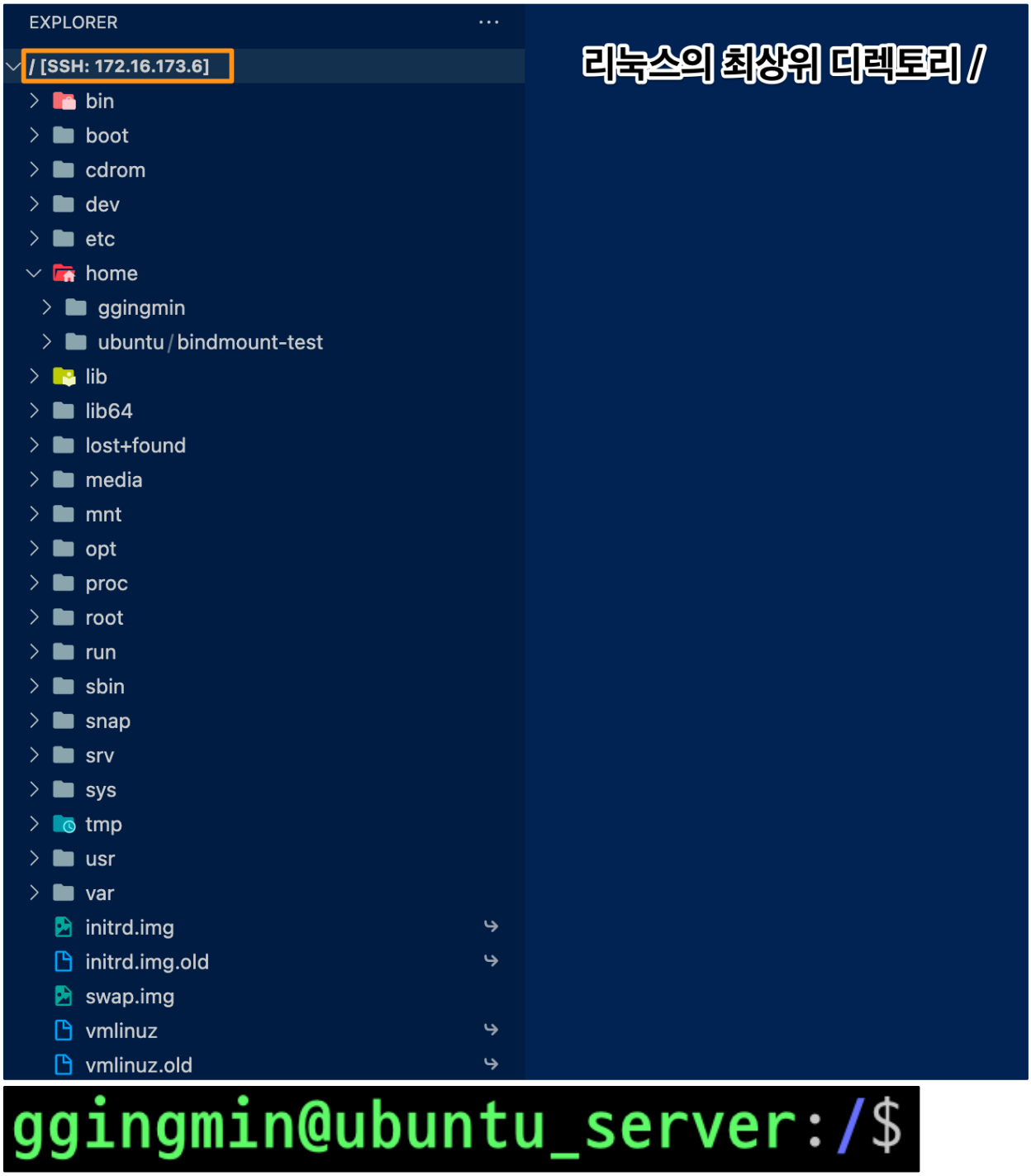
1. sh - Bourne Shell
    - Bourne Shell 은 벨 연구소의 스티븐 본이 1977년 발표한 유닉스 Shell 입니다. 줄여서 sh 이라고도 합니다. Bash Shell 이 개발되기 전까지 광범위하게 사용되었습니다.
2. bash - Bourne-again Shell
    - Bourne-again Shell 은 브라이언 폭스가 1989년 발표하였으며, 리눅스의 여러 배포판과 macOS 의 기본 Shell 로 세팅되어 사용 되었습니다.(macOS 10.15 Catalina 이후 기본 Shell 이 zsh 로 변경되었습니다.)
3. zsh - Z Shell
    - 현재 macOS의 기본 Shell 로 사용되고 있는 zsh 은 1990년 피터 팔스타드가 발표하였습니다. 여러 Shell 과 비교했을 때 많은 점이 개선 되었죠. 자동완성 속도가 향상되었고 다양한 와일드카드 템플릿이 추가되었습니다.

부록에서는 다양한 Shell 의 종류 중 bash 에 대하여 다루겠습니다.

1. 파일 탐색

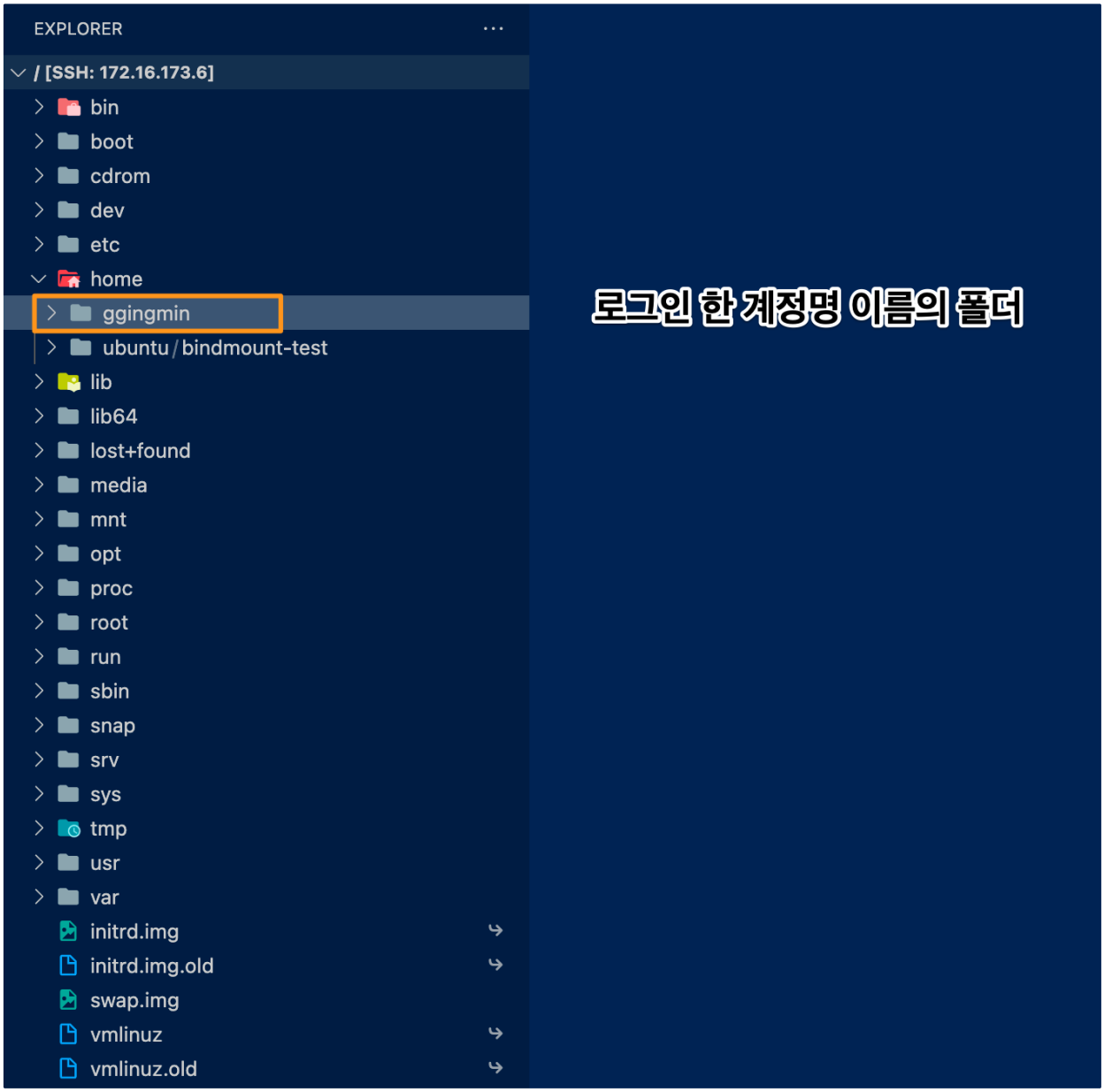
Shell 에서 작업을 할 때 가장 막막한 것이 바로 파일을 찾고 폴더 간 이동을 하는 것입니다. 아래는 Ubuntu 의 기본 디렉토리 구조를 최상위 디렉토리인 루트에서 바라본 파일트리입니다.

루트 디렉토리는 Shell 에서 `/` 로 표기합니다.



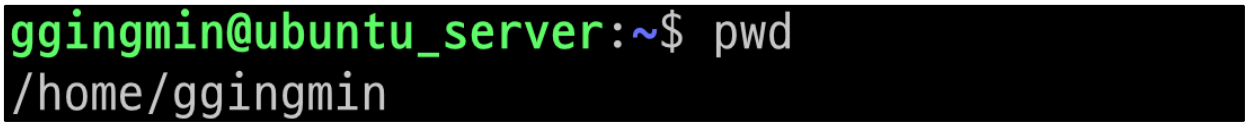
일반적으로 Ubuntu 가 부팅되고 난 후에 프롬프트가 시작되는 지점은 루트가 아닌 `/home` 디렉토리 하위의 로그인한 계정명으로 되어있는 디렉토리입니다.

이를 홈 디렉토리라고 하며, `~` 형태로 표시됩니다.



1) `pwd` - 현재 위치한 디렉토리 확인

```
$ pwd
```



`pwd` 는 **pass working directory** 의 줄임말로 working directory, 즉 현재 프롬프트가 위치한 디렉토리를 출력합니다.

2) `cd` - 디렉토리 변경

```
$ cd # 홈 디렉토리로 이동 $ cd / # 루트 디렉토리로 이동 $ cd /home # /home 으로 이동
```

```
ggingmin@ubuntu_server:~$ cd /home
ggingmin@ubuntu_server:/home$
```

`cd` 는 **change directory** 의 줄임말로 다른 디렉토리로 이동을 하기 위해 사용됩니다. cd 뒤에는 경로를 명시해야 하며, 특별한 경로를 명시하지 않으면 계정의 홈 디렉토리 `~` 로 이동하게 됩니다.

> 💡 **절대경로(Absolute Path) VS 상대경로(Relative Path)**
>
> 절대경로는 현재 위치한 디렉토리와 상관없는 완전한 형태의 경로를 의미합니다.
> `pwd` 명령어를 통해 조회되는 것이 바로 절대경로 입니다.
>
> 상대경로는 현재 위치를 기준으로 경로를 설정한 것으로 다음의 기호를 사용하여 나타냅니다.
> - `.` : 현재 디렉토리
> - `..` : 상위 디렉토리
>
> 가령, 홈 디렉토리 `~` 에서 루트 디렉토리로 가는 방법은 다음과 같습니다.
> `~$ cd ../..`
> `~$ cd /`

3) `ls` - 파일 및 디렉토리 조회

```
$ ls # 현재 디렉토리 기준 조회 $ ls -l # 상세 정보 조회 $ ls -a # 숨김 파일까지 조회 $ ls -al # -l과 -a 옵션을 모두 적용 $ ls / # 루트 디렉토리 조회 $ ls [절대경로/상대경로] # 입력한 경로 조회
```

```
ggingmin@ubuntu_server:~$ ls
apache.tar  chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  redis  test  tmp
```

```
-rw-rw-r-- 1 ggingmin ggingmin 138387456 Aug 20 07:27 apache.tar
drwxrwxr-x 8 ggingmin ggingmin      4096 Sep  1 05:52 chapter-2
drwxrwxr-x 5 ggingmin ggingmin      4096 Jul  2 03:40 docker
-rw-rw-r-- 1 ggingmin ggingmin  69774135 Jul 14 06:14 minikube-linux-amd64
drwxrwxr-x 2 ggingmin ggingmin      4096 Aug 25 08:12 node-project
drwxrwxr-x 3 ggingmin ggingmin      4096 Aug 19 06:31 portfolio
drwxrwxr-x 3 ggingmin ggingmin      4096 Sep  2 08:29 practice
drwxrwxr-x 2 ggingmin ggingmin      4096 Aug 20 06:54 redis
drwxrwxr-x 2 ggingmin ggingmin      4096 Sep  1 06:42 test
drwxrwxr-x 3 ggingmin ggingmin      4096 Jun 22 06:05 tmp
```

권한 **소유자** **그룹** **용량** **최종수정일**

```
ggingmin@ubuntu_server:~$ ls -a
.              .bash_logout  .config     .local                .node_repl_history  .profile                test
..             .bashrc       docker      .minikube             .npm                .python_history         tmp
apache.tar     .cache        .gitconfig  minikube-linux-amd64  portfolio           redis                   .vscode-server
.bash_history  chapter-2     .gnupg      node-project          practice            .sudo_as_admin_successful  .wget-hsts
```

`ls` 는 파일 및 디렉토리를 조회하는 데에 사용됩니다. 옵션에 따라서 다양한 결과가 출력되는데요, `-l` 과 `-a` 옵션은 모두 빈번하게 쓰이고 있습니다. 리눅스에서는 `.bashrc` 와 같은 숨김 파일을 수정하는 일이 많은데 이 때 `-a` 옵션이 유용하게 사용됩니다.

`ls` 의 인수로 경로를 넘겨주게 되면 해당 경로의 파일과 디렉토리 정보를 출력합니다.

> 💡 **권한 정보**
> 파일 조회 시 권한을 나타내는 문자열을 4개의 부분으로 분할 할 수 있습니다.
>
> 예시 : `drwxrwxr-x`
>
> 1. 첫 번째 글자
> - `d` 혹은 `-` 로 시작합니다. `d` 는 디렉토리를, `-` 는 일반적인 파일을 의미합니다.
>
> 2. 두 번째 ~ 네 번째 글자
> - 파일 소유자의 권한을 나타냅니다. `r` 은 read(읽기), `w` 는 write(쓰기), `x` 는 execution(실행)을 각각 나타냅니다.
>
> 3. 다섯 번째 ~ 일곱 번째 글자
> - 파일 그룹의 권한을 나타냅니다.
>
> 4. 여덟 번째 ~ 열 번째 글자
> - 모든 사용자의 권한을 나타냅니다.

2. 파일 조작

파일을 복사하거나 삭제하는 일, 또는 디렉토리를 만드는 일도 거의 매일 사용하는 기능입니다. 이를 효율적으로 사용하기 위해서는 **와일드카드**를 먼저 익히는 것이 좋습니다. 특정 파일을 한꺼번에 가리킬 때 굉장히 유용한 기능이죠.

- `*` : **하나의 문자 혹은 문자열이 어떤 것이라도 포함**
  - 단독으로 `*` 라고 쓰면 모든 파일을 가리킵니다.
  - `Docker*` 라고 작성하면 `Dockerfile` 과 `Dockerfile.base` 을 모두 포함합니다.
- `?` : **문자의 개수가 일치해야 포함**
  - `Docker????` 이라고 작성하면 `Dockerfile` 은 포함되지만 `Dockerfile.base` 은 포함되지 않습니다.
- `[(문자열)]*` : **대괄호 안의 각 문자로 시작하는 파일을 가리킵니다.**
  - `[xyz]*` 라고 쓰면 `x`, `y`, `z` 로 시작하는 모든 파일이 해당됩니다.
- `*[(문자열)]` : **대괄호 안의 각 문자로 끝나는 파일을 가리킵니다.**
  - `*[xyz]` 라고 쓰면 `x`, `y`, `z` 로 끝나는 모든 파일이 해당됩니다.

1) `cp` - 파일 복사

```
~$ cp [원본 파일명] [복사되는 파일 명] ~$ cp apache.tar apache2.tar ~$ cp -i
apache.tar apache2.tar # 동일한 파일명 존재시 사용자에게 먼저 묻기 ~$ cp [원본 파일명]...
[복사 대상 디렉토리] ~$ cp apache.tar apache2.tar ./practice ~$ cp -R [원본 디렉토리]
[복사 대상 디렉토리] ~$ cp -R ./practice ./practice2
```

```
ggingmin@ubuntu_server:~$ cp apache.tar apache2.tar
ggingmin@ubuntu_server:~$ ll
total 338576
drwxr-xr-x 17 ggingmin ggingmin      4096 Sep  2 10:04 ./
drwxr-xr-x  4 root     root          4096 Aug 18 15:21 ../
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:04 apache2.tar
-rw-rw-r--  1 ggingmin ggingmin 138387456 Aug 20 07:27 apache.tar
```

```
ggingmin@ubuntu_server:~$ cp -i apache.tar apache2.tar
cp: overwrite 'apache2.tar'?
```

```
ggingmin@ubuntu_server:~$ cp apache.tar apache2.tar ./practice
ggingmin@ubuntu_server:~$ ls ./practice
apache2.tar  apache.tar  dev
```

```
ggingmin@ubuntu_server:~$ cp -R ./practice ./practice2
ggingmin@ubuntu_server:~$ ls
apache2.tar  chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  practice2  redis  test  tmp
```

파일 복사는 `cp` 명령어를 사용합니다. 두번째 인자로 전달되는 것이 복사되어 생성되는 파일입니다. 만약에 동일한 파일명을 가진 파일이 이미 존재한다면 경고 없이 덮어쓰기 되기 때문에 반드시 주의해야 합니다.

위와 같은 상황을 방지하려면 `-i` 옵션을 추가하면 됩니다. 동일한 파일명이 존재할 경우 경고가 뜨기 때문에 불상사를 막을 수 있죠

여러 파일을 한꺼번에 특정 디렉토리에 복사하는 것도 가능합니다. 파일명을 차례대로 나열하고 마지막 인수로 디렉토리를 넘겨주면 됩니다.

만약 파일이 아닌 디렉토리 단위로 복사를 해야한다면 `-R` 옵션을 적용하면 됩니다. 이 역시 동일한 디렉토리가 존재하는 경우 덮어쓰기 되기 때문에 유의해야 합니다.

2) `mv` - 파일 이름 바꾸기 및 파일 이동

```
~$ mv [원본 파일명] [복사되는 파일 명] ~$ mv apache.tar apache3.tar ~$ mv -i
apache2.tar apache3.tar # 동일한 파일명 존재시 사용자에게 먼저 묻기 ~$ mv [원본 파일명]...
[복사 대상 디렉토리] ~$ mv apache3.tar ./practice ~$ mv [원본 디렉토리] [복사 대상 디렉토
리] ~$ mv ./practice2 ./practice3
```

```
ggingmin@ubuntu_server:~$ mv apache.tar apache3.tar
ggingmin@ubuntu_server:~$ ll
total 338580
drwxr-xr-x 18 ggingmin ggingmin      4096 Sep  2 10:22 ./
drwxr-xr-x  4 root     root          4096 Aug 18 15:21 ../
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:09 apache2.tar
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:09 apache3.tar
```

```
ggingmin@ubuntu_server:~$ mv -i apache2.tar apache3.tar
mv: overwrite 'apache3.tar'?
```

```
ggingmin@ubuntu_server:~$ mv apache3.tar ./practice
ggingmin@ubuntu_server:~$ ll ./practice
total 405452
drwxrwxr-x  3 ggingmin ggingmin      4096 Sep  2 10:24 ./
drwxr-xr-x 18 ggingmin ggingmin      4096 Sep  2 10:24 ../
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:10 apache2.tar
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:09 apache3.tar
-rw-rw-r--  1 ggingmin ggingmin 138387456 Sep  2 10:10 apache.tar
drwxrwxr-x  3 ggingmin ggingmin      4096 Sep  2 10:08 dev/
```

```
ggingmin@ubuntu_server:~$ mv ./practice2 ./practice3
ggingmin@ubuntu_server:~$ ls
apache2.tar  chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  practice3  redis  test  tmp
```

`mv` 명령어는 파일 이동 뿐만 아니라 파일 혹은 디렉토리의 이름을 변경하는 데에도 사용합니다. `-i` 옵션을 주어 동일한 파일명이 존재할 때 덮어 씌워지는 것을 방지할 수 있습니다.

3) `rm` - 파일 삭제

```
~$ rm [파일명]... ~$ rm apache2.tar ~$ rm -i apache2.tar apache3.tar # 파일 삭제하기 전에 사용자에게 묻기 ~$ rm -r [디렉토리명]... ~$ rm -r ./practice3
```

```
ggingmin@ubuntu_server:~$ rm apache2.tar
ggingmin@ubuntu_server:~$ ls
chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  practice2  redis  test  tmp
```

```
ggingmin@ubuntu_server:~$ rm -r ./practice3
ggingmin@ubuntu_server:~$ ls
chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  redis  test  tmp
```

파일 삭제는 `rm` 을 사용하며, 디렉토리를 삭제할 때에는 `-r` 옵션을 추가해주어야 합니다.

4) `mkdir` - 디렉토리 생성

```
~$ mkdir [디렉토리명]... ~$ mkdir ./practice4 ./practice5
```

```
ggingmin@ubuntu_server:~$ mkdir ./practice4 ./practice5
ggingmin@ubuntu_server:~$ ls
chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  practice4  practice5  redis  test  tmp
```

3. 확장

Shell 은 다양한 확장 기능을 제공합니다. 확장을 통해 시스템 내부에 지정된 변수의 값이나 연산을 실행해서 그 결과를 사용자에게 반환합니다. 본격적으로 확장 기능을 알아보기 전에 문자열을 Shell 에 출력하는 명령어인 `echo` 를 익혀보겠습니다.

```
~$ echo "Hello, Shell!"
```

```
ggingmin@ubuntu_server:~$ echo "Hello, Shell!"
Hello, Shell!
```

인수로 문자열을 넘겨주면 문자열을 그대로 Shell에 출력해줍니다.

3.1 경로 확장

```
~$ ls prac*
```

```
ggingmin@ubuntu_server:~$ ls prac*
practice:
apache2.tar  apache3.tar  apache.tar  dev

practice4:

practice5:
```

`prac` 이라는 문자열로 시작하는 모든 디렉토리명을 조회하였습니다.

```
~$ ls /home/*
```

```
ggingmin@ubuntu_server:~$ ls /home/*
/home/ggingmin:
chapter-2  docker  minikube-linux-amd64  node-project  portfolio  practice  practice4  practice5  redis  test  tmp

/home/ubuntu:
bindmount-test
```

와일드 카드는 위와 같이 경로의 일부로 사용할 수도 있습니다.

3.2 틸드 확장

```
~$ echo ~
```

```
ggingmin@ubuntu_server:~$ echo ~
/home/ggingmin
```

`~` 틸드는 물결표라는 뜻으로, 리눅스에서는 홈 디렉토리를 나타냅니다.

3.3 산술 확장

```
~$ echo $((2 + 3))
```

```
ggingmin@ubuntu_server:~$ echo $((2+3))
5
```

`$((산술식))` 의 형태로 작성하면 문자열이 아니라 내부의 산술식을 연산한 결과를 반환합니다.

3.4 중괄호 확장

```
~$ echo INF{P,J}
```

```
ggingmin@ubuntu_server:~$ echo INF{P,J}
INFP INFJ
```

중괄호 확장은 반복작업을 줄이는 데에 큰 도움을 줍니다. 정해진 키워드를 문자열의 특정 위치에 넣을 때 많이 쓰입니다. 예시를 좀 더 보겠습니다.

```
~$ echo {A..Z} ~$ echo {1..100}
```

```
ggingmin@ubuntu_server:~$ echo {A..Z}
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ggingmin@ubuntu_server:~$ echo {1..100}
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 4
8 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
 93 94 95 96 97 98 99 100
```

연속적인 값을 `..` 으로 연결하면 표기된 값과 더불어 사이에 위치한 모든 값을 반환합니다. 이러한 특성을 활용하면 연월일을 순차로 생성하는 작업을 매우 손쉽게 할 수 있습니다.

```
~$ mkdir weblog ~$ cd weblog ~$ mkdir {2021..2022}-{01..12}
```

```
ggingmin@ubuntu_server:~$ mkdir weblog
ggingmin@ubuntu_server:~$ cd weblog/
ggingmin@ubuntu_server:~/weblog$ mkdir {2021..2022}-{01..12}
ggingmin@ubuntu_server:~/weblog$ ls
2021-01  2021-03  2021-05  2021-07  2021-09  2021-11  2022-01  2022-03  2022-05  2022-07  2022-09  2022-11
2021-02  2021-04  2021-06  2021-08  2021-10  2021-12  2022-02  2022-04  2022-06  2022-08  2022-10  2022-12
```

3.5 매개변수 확장

```
~$ echo $USER
```

```
ggingmin@ubuntu_server:~$ echo $USER
ggingmin
```

`USER` 에는 시스템 내부적으로 사용자명이 저장되어 있습니다. 이렇게 시스템에서 사용되는 특정 값들을 대입한 것을 환경변수라고 합니다. 이를 사용하기 위해서는 단독으로 `USER` 를 입력하는 것이 아니라 `$` 표시를 앞에 표기해주어야 합니다.

3.6 명령어 치환

```
~$ echo $(ls)
```

```
ggingmin@ubuntu_server:~$ echo $(ls)
chapter-2 docker minikube-linux-amd64 node-project portfolio practice practice4 practice5 redis test tmp weblog
```

명령어 치환은 쉽게 말해 명령어 내부에서 다른 명령어를 실행한 값을 다시 활용하는 것을 가리킵니다. `ls` 명령어를 통해 반환된 값이 다시 `echo` 명령어의 인수로 사용되었기 때문에 위와 같은 결과가 출력됩니다.

3.7 큰따옴표 표기

```
~$ mkdir test dir ~$ mkdir "test dir"
```

```
ggingmin@ubuntu_server:~$ ls
 chapter-2   minikube-linux-amd64   portfolio   practice4   redis  'test dir'   weblog
 docker      node-project           practice    practice5   test    tmp
```

만약 파일이나 디렉토리 명에 공백이 있다면 반드시 큰 따옴표로 감싸서 작성해주어야 합니다. 컴퓨터에게 이 문자열이 하나의 인수라는 것을 알려주는 것이죠.

```
~$ echo $(ls)
```

```
ggingmin@ubuntu_server:~$ echo "$(ls)"
chapter-2
docker
minikube-linux-amd64
node-project
portfolio
practice
practice4
practice5
redis
test
test dir
tmp
weblog
```

큰따옴표를 감싸더라도 명령어 치환, 매개변수 확장, 산술 확장 등은 정상적으로 작동합니다. 또한, 줄바꿈 문자나 공백 문자도 해석하여 출력해줍니다.

3.8 작은따옴표 표기

```
~$ echo '$USER $((2+3)) $(ls)'
```

```
ggingmin@ubuntu_server:~$ echo '$USER $((2+3)) $(ls)'
$USER $((2+3)) $(ls)
ggingmin@ubuntu_server:~$ echo $USER $((2+3)) $(ls)
ggingmin 5 chapter-2 docker minikube-linux-amd64 node-project portfolio practice practice4 practice5 redis test test dir tmp weblog
```

작은따옴표는 모든 확장을 무시하고 순수하게 문자열을 반환합니다.

4. 권한

4.1 접근 권한 조작

스택오버플로우나 각종 개발 문서를 보면 `chmod` 라는 키워드가 굉장히 많이 등장합니다. **change mode** 의 약자로, 리눅스 환경에서 파일이나 디렉토리의 접근 권한을 조작하는 작업을 수행합니다. 접근 권한은 읽기(read), 쓰기(write), 실행(execution)으로 구성되어 있습니다. 함께 쓰이는 값을 다음과 같이 살펴보겠습니다.

4.1.1 파일 접근 권한

```
~$ chmod 777 [파일명]... ~$ chmod 755 [파일명]... ~$ chmod 700 [파일명]... ~$ chmod 666 [파일명]... ~$ chmod 644 [파일명]... ~$ chmod 600 [파일명]...
```

- `chmod 777` : 읽기, 쓰기, 실행에 아무 제약이 없습니다.
- `chmod 755` : 파일 소유자는 읽기, 쓰기, 실행을 모두 수행할 수 있으며, 소유자 외에는 읽기와 실행만 가능합니다. 대부분의 프로그램이 이 권한을 가지고 있습니다.
- `chmod 700` : 파일 소유자만 읽기, 쓰기, 실행을 수행할 수 있고 그 외에는 아무 권한을 가지지 못합니다.
- `chmod 666` : 모든 사용자는 읽기, 쓰기 권한을 가집니다.
- `chmod 644` : 소유자는 읽기, 쓰기 권한을 가지고 그 외에는 읽기만 가능합니다.
- `chmod 600` : 소유자만 읽기, 쓰기 권한을 가지고 그 외에는 아무 권한을 가지지 못합니다.

4.1.2 디렉토리 접근 권한

```
~$ chmod 777 [디렉토리명]... ~$ chmod 755 [디렉토리명]... ~$ chmod 700 [디렉토리명]...
```

💡 **디렉토리 관점의 r, w, x**
`r` : 디렉토리 안의 파일과 디렉토리 목록을 조회할 수 있는 권한
`w` : 디렉토리 안의 파일과 디렉토리를 수정, 삭제하거나 새로운 파일, 디렉토리를 생성하는 권한
`x` : 디렉토리 내에 진입할 수 있는 권한

- `chmod 777` : `r`, `w`, `x` 에 아무 제약이 없습니다
- `chmod 755` : 파일 소유자는 `r`, `w`, `x` 를 모두 수행할 수 있으며, 소유자 외에는 `r`, `w` 만 가능합니다.

- `chmod 700` : 파일 소유자만 `r`, `w`, `x` 를 수행할 수 있고 그 외에는 아무 권한을 가지지 못합니다.

4.2 루트 계정 접근 및 루트 패스워드 설정

```
ggingmin@ubuntu_server:~$ sudo su root@ubuntu_server:/home/ggingmin# passwd
```

```
ggingmin@ubuntu_server:~$ sudo su
root@ubuntu_server:/home/ggingmin# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@ubuntu_server:/home/ggingmin# exit
ggingmin@ubuntu_server:~$
```

특정 작업을 위해 루트 계정으로 사용자를 바꿔야 할 때가 있습니다. 그 때 사용하는 것이 `su` 명령입니다. **substitute user** 의 약자로, 현재 접속해있는 사용자를 바꿔야 하는 경우에 사용합니다. `su - 사용자명` 의 형태로 사용자를 명시하지 않고 `su` 만 사용하게 되면 루트 계정으로 전환을 시도합니다. 이렇게 루트 계정으로 전환하기 위해서는 루트 권한으로 실행을 해주어야 하는데, 여기서 추가되는 명령이 바로 `sudo` 입니다.

루트 계정의 패스워드를 설정하기 위해서는 먼저 루트 계정으로 접근한 뒤 `passwd` 명령어를 실행하여 설정하면 됩니다.

4.3 소유자 변경

```
~$ sudo chown root ./practice5
```

```
drwxrwxr-x  2 root     ggingmin     4096 Sep  2 10:32  practice5/
```

`chown` 은 파일 혹은 디렉토리의 소유자를 변경하기 위한 명령어 입니다. 위 명령어를 실행하면 이미지와 같이 `root` 로 소유자가 변경된 것을 확인할 수 있습니다.