데이터사이언스를위한 지식관리시스템

2주차: 리눅스 터미널 주요 커맨드 살펴보기

2주차 목표

- 터미널 화면에 익숙해지기, 터미널 화면을 탐색기나 Finder처럼 다루기
- 연구에 필요한 서버를 사용하게 된다면 이 환경에서 시작할 확률이 높습니다
- 그리고 이 과정은 3, 4주차 이후의 내용을 이해하는 기반 지식이 될 것입니다

- 이러한 순서로 진행합니다
  - 터미널에 접속하면 나오는 커맨드라인이 무엇인지 알아봅니다
  - 커맨드라인을 구성하는 프롬프트는 무엇인지 알아봅니다
  - 프롬프트를 구성하는 각 요소의 의미는 무엇인지 알아봅니다
  - 그리고 이들을 잘 핸들링 할 수 있는 유용한 커맨드들을 알아봅니다

터미널은 무엇이고 커맨드는 무엇인가요

- 터미널: 텍스트 환경을 베이스로 하여 사용자와 대화형 커맨드 라인 인터페이스를 제공하는 프로그램

- 커맨드: 운영체제에 전달하고자 하는 명령, 터미널의 내장 명령어 외에 다른 프로그램의 실행을 명령할 수도 있음

- 텍스트상에서 펼쳐지는 탐색기, Finder 같은 것이라고 생각해도 되겠습니다
- 그래서 터미널을 열고 커맨드를 학습하겠습니다

내장 명령어와 다른 프로그램

- 터미널의 프로그램 위치는 /bin/bash
- /bin/bash의 의미: 루트 디렉토리 밑에 bin 이라는 디렉토리 밑에 bash 라는 파일이 터미널 프로그램 입니다

- 이 bash라는 프로그램이 자체적으로 가지고 있는 명령어가 내장 명령어 입니다
- 그렇지 않은 것은 터미널 위에서 실행되는 다른 프로그램 입니다
- 마치 탐색기, Finder 자체의 기능인지 아니면 그 위에서 실행되는 다른 프로그램인지 구분하는 것과 같습니다

- 이 수업에서는 이 둘을 별도로 구분하지 않고 그냥 합쳐서 커맨드라고 하겠습니다

왜 터미널은 새까맣거나 새하얗거나 둘 중 하나일지 생각해보겠습니다

- GUI(Graphic User Interface) 라는 용어는 들어 보셨을 듯 합니다
- CLI(Command Line Interface), 이것이 터미널 환경을 지칭하는 용어입니다

- 텍스트 기반

- 우리가 흰 종이 위에 검은 글씨를 쓰듯이 새까만 화면에 하얀 글자
- 하얀 바탕, 검정 바탕 보다는 텍스트 기반의 대화형 인터페이스라는 것을 생각합시다

프롬프트

- 많이 익숙한 말이지만 터미널에서의 프롬프트는 아래와 같은 의미를 갖습니다
- 커맨드 입력을 기다리는 상태를 알리는 메시지
- $ 또는 # 가 많이 쓰입니다

- 그 외의 문자들은 부가적으로 아래의 의미를 표시합니다
- 현재 사용중인 계정: Ubuntu
- 현재의 작업 디렉토리: /home/ubuntu
- 합치면: ubuntu@ubuntu-server:/home/ubuntu$

- 앞에 ubuntu는 무엇이고 뒤의 ubuntu-server는 무엇인지 알아보겠습니다

프롬프트의 부가정보는 결국 나는 누구? 여긴 어디? 에 대한 답이 되겠습니다

- 엑셀 작업할 때에도 작업 디렉토리가 있듯이 터미널에서도 작업 디렉토리가 있습니다
- Unix-Like 운영체제의 파일 경로를 다시 생각해봅시다
- 루트 파일시스템이 있고 그 아래에 디렉토리들이 있습니다

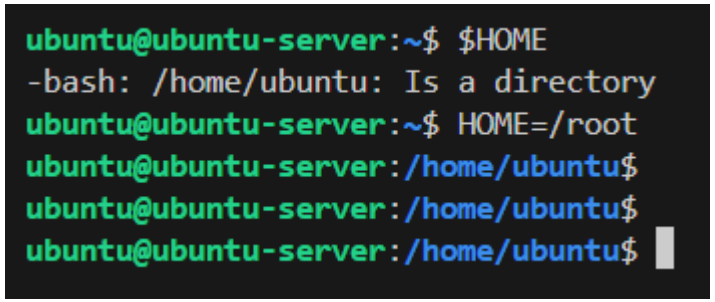
- ~ 는 현재 사용중인 계정의 홈 디렉토리를 의미합니다
- ubuntu@ubuntu-server:~/$ 이라고 나오면 그 의미는 무엇일까요?
- ubuntu-server 라고 하는 hostname을 가진 PC의 ubuntu계정으로 로그인했는데 현재의 위치는 ubuntu계정의 home 디렉토리
- ubuntu@ubuntu-server:/home/ubuntu$ 와 동일한 뜻입니다

```
ubuntu@ubuntu-server:~$ cd /home/ubuntu/
ubuntu@ubuntu-server:~$
ubuntu@ubuntu-server:~$
```

디렉토리 이동 커맨드를 입력했는데 제자리

Home 디렉토리는 누가 정해주나요? => 환경변수

- 환경변수는 MS Windows에도 있습니다
- 현재 로그인된 계정의 환경변수를 보는 방법: env
- 특정 환경변수를 보는 방법 예를 들면 HOME 환경변수를 보려면 $HOME
- 이 중에서 HOME 이라는 환경변수가 결정해줍니다
- 이 환경변수를 바꾸면?
- HOME=/root 라고 하면 /root 가 home 디렉토리가 됩니다
- 그러면 /home/ubuntu 가 더 이상 ~ 으로 표시되지 않습니다
- /home/ubuntu는 더 이상 HOME이 아니기 때문입니다

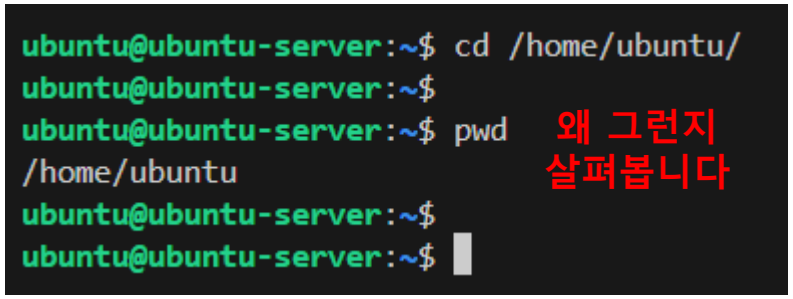


갑자기 여기서 디렉토리 경로가 파란색으로 나오는 이유는 무엇일까요?

여기가 어딘지 확실하게 알고 싶을때

- ~ 부터 나오니까 도대체 여기가 어딘지 모르겠습니다
- 그럴때는 pwd 명령을 실행해봅니다

- 그러면 왜 도대체 여기가 어딘지 한번에 안 알려주는 것인가 하는 의문이 들 수 있습니다
- Home 경로를 생략함으로써 전체 경로가 짧게 서술되는 장점이 있습니다



디렉토리 이동 커맨드를 입력했는데 역시나 제자리

절대경로와 상대경로

- 이것만 기억해주세요
- / 부터 시작하면 절대경로
- 아니면 다 상대경로
- 왜냐면?
- 루트 디렉토리가 / 이니까
- 루트 디렉토리부터 서술하면? 절대경로
- 루트 디렉토리를 모르면? 상대경로
- ~ 부터 시작하면 누구에 대한 상대경로? HOME 디렉토리에 대한 상대경로

. (오타 아니고 dot 맞습니다)

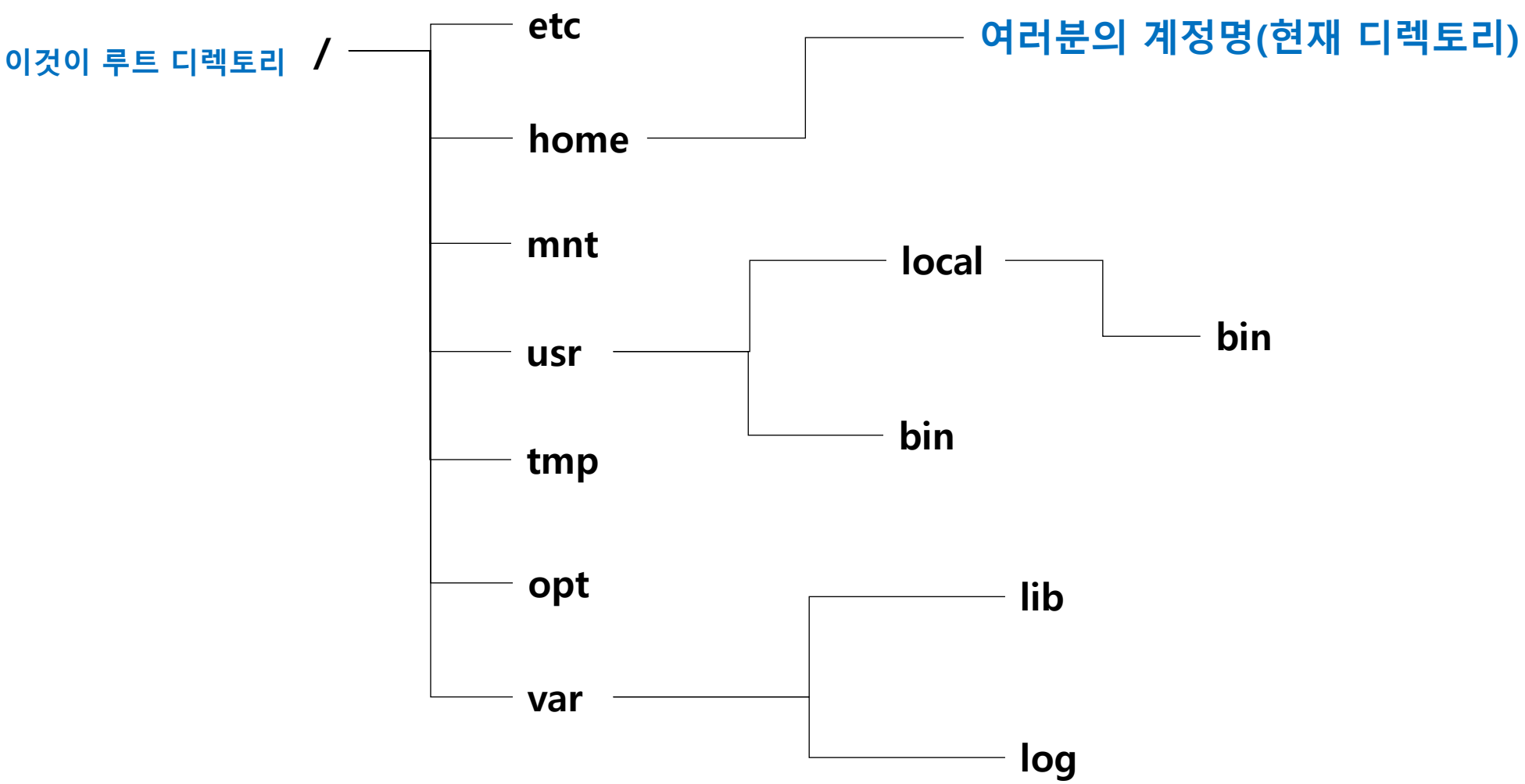
- 현재 디렉토리를 의미합니다
- 아주 중요
- 아래 둘의 차이를 구분해봅시다
- /home/ubuntu, 무조건 /home/ubuntu, 절대경로
- ./home/ubuntu, 현재 디렉토리가 /home/ubuntu에 있다면 /home/ubuntu/home/ubuntu, 상대경로
- 왜 상대경로인가요? / 으로 시작 하는게 아니라 . 으로 시작 하니까요

- 참고로 . 말고 .. 은 상위 디렉토리 입니다
- 예를 들어 /home/ubuntu 위치에서 .. 은 /home

터미널에서 finder나 탐색기를

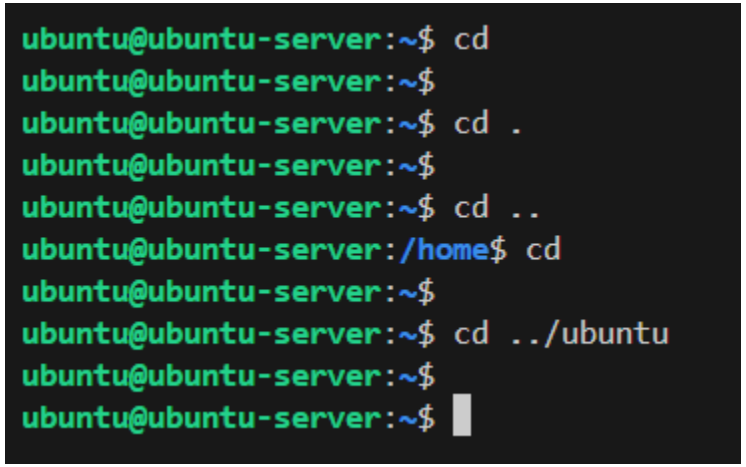
- 아마도 이 수업시간에서 가장 많이 쓸 커맨드 cd 와 ls
- 폴더를 클릭하면 디렉토리를 이동하듯이 cd를 사용하고 디렉토리를 이동하면 파일 내용이 보이듯이 ls를 써보겠습니다
- 그리고 더블 클릭하여 실행하듯이 터미널에서도 파일을 실행해보겠습니다

Ubuntu 22.04 server 기준 주요 디렉토리 일부 구조



이를 바탕으로 디렉토리 이동을 해봅니다

- cd <이동할 디렉토리>
- cd . => 현재 디렉토리로 이동... = 제자리 (?)
- cd .. => 상위 디렉토리로 이동
- cd ../ubuntu => 상위 디렉토리 아래의 ubuntu로 이동.....(???)
- cd, 그냥 cd만 => home 디렉토리로 이동



상위 디렉토리로 이동했더니 /home이 나온 이유를 생각해봅시다

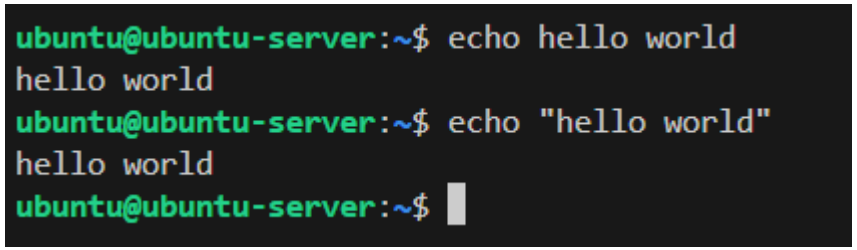
상위 디렉토리로 갔다가 그 아래의 ubuntu 디렉토리로

결국 제자리

이제부터 argument에 대해 논할 시간입니다

- argument: 매개변수, 무언가 실행되는 것에게 변수를 전달하는 방법
- 변수에 따라 프로그램의 동작에 옵션을 부여하는 방법
- 이 방법은 커맨드라인 뿐만이 아니라 앞으로 프로그래밍에서도 많이 사용하게 될 것입니다

- 그런 의미로 hello world 해보겠습니다
  - echo hello world
  - echo “hello world”
  - 무슨 차이일까요?
  - 매개변수가 1개 이냐 2개 이냐의 차이 입니다
  - echo 라는 프로그램은 복수의 매개변수를 받으며 매개변수를 모두 화면 출력 합니다



cd 명령도 결국 argument를 사용합니다

- 폴더 이동, 즉 작업 디렉토리의 이동을 위해서 cd를 사용하는데 argument가 필요합니다
- 탐색기, finder에서 파일 옆에 시간이 나오고 정렬을 하듯이 우리도 그렇게 해보겠습니다
- 이를 위해서는 ls 뒤에 argument가 필요합니다
- ls -l 에서 argument는 -l 입니다

cd 명령의 매개변수가 /home/ubuntu

```
ubuntu@ubuntu-server:~$ cd /home/ubuntu/
ubuntu@ubuntu-server:~$ ls
ubuntu@ubuntu-server:~$ ls -al
total 32
drwxr-x--- 4 ubuntu ubuntu 4096 Sep  1 14:18 .
drwxr-xr-x 3 root   root   4096 Sep  1 14:07 ..
-rw------- 1 ubuntu ubuntu  159 Sep  4 15:26 .bash_history
-rw-r--r-- 1 ubuntu ubuntu  220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Jan  6  2022 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Sep  1 14:07 .cache
-rw-r--r-- 1 ubuntu ubuntu  807 Jan  6  2022 .profile
drwx------ 2 ubuntu ubuntu 4096 Sep  1 14:08 .ssh
-rw-r--r-- 1 ubuntu ubuntu    0 Sep  1 14:18 .sudo_as_admin_successful
ubuntu@ubuntu-server:~$
```

매개변수가 있는 ls

매개변수가 없는 ls

ls

- 지정된 디렉토리내에 있는 디렉토리와 파일을 리스트업
- 기본적으로 숨김 파일, 디렉토리는 보여주지 않습니다
- . 으로 시작하는 파일, 디렉토리는 숨김처리 됩니다
- ls 한번 해보고 ls / 해보겠습니다

ls 했더니 아무것도 없습니다
현재 디렉토리에는 숨김 처리 되지 않은 파일, 디렉토리가 없다는 의미입니다

```
ubuntu@ubuntu-server:~$ ls
ubuntu@ubuntu-server:~$ ls /
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  swap.img  sys  tmp  usr  var
ubuntu@ubuntu-server:~$
```

루트 디렉토리 밑에는 이러한 디렉토리와 파일이 있습니다
마치 MS Windows 에서 C:₩ 드라이브를 열어본것과 비슷합니다

ls

- ls의 다양한 옵션 중에서 아래 3개만 확인하겠습니다
- 먼저 ls 도 해보고 ll도 해보겠습니다
- ll = ls -alF
- ll이 왜 ls -alF 인지는 2주차 종료 직전에 확인합니다
- a: 숨김파일, 디렉토리도 출력
- l: 파일, 디렉토리 상세 정보와 함께 리스트로 출력
- F: 파일 속성 출력

각 커맨드들이 어떠한 옵션이 있는지 알 수 있는 방법

- 검색
- man
- 그리고 --help

```
ubuntu@ubuntu-server:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic characters
      --block-size=SIZE      with -l, scale sizes by SIZE when printing them;
                               e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of last
                               modification of file status information);
```

ls --help 를 통해 알 수 있는 도움말

```
LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory

       Mandatory arguments to long options are mandatory for sh

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file
```

man ls를 통해 알 수 있는 도움말

중간정리

- 프롬프트에 나오는 요소: 계정, 호스트명, 디렉토리 경로
- 디렉토리 경로는 절대경로와 상대경로
- 디렉토리를 이동하는 방법, 디렉토리의 내용을 보는 방법
- 마치 텍스트 대화로 탐색기나 Finder를 이용하는 것과 같은 느낌
- 우리가 연구용 리눅스 서버에 접속하면 그런 텍스트 화면을 볼 확률이 높습니다
- 그래서 리눅스 터미널에서 기본적인 파일 탐색 방법을 알아보았습니다
- 지금부터는 리눅스 터미널에서 세부적인 파일 탐색 방법을 알아보겠습니다

find

- 지정된 디렉토리를 기준으로 하위 디렉토리와 파일을 리스트업
- 기본적으로 모든 디렉토리와 파일을 리스트업
- 옵션에 따라 필터링 가능

```
ubuntu@ubuntu-server:~$ ls
ubuntu@ubuntu-server:~$
ubuntu@ubuntu-server:~$ find
.
./.sudo_as_admin_successful
./.bash_history
./.profile
./.bashrc
./.cache
./.cache/motd.legal-displayed
./.bash_logout
./.ssh
./.ssh/authorized_keys
ubuntu@ubuntu-server:~$
ubuntu@ubuntu-server:~$
```

ls 에서는 아무것도 나오지 않다가

find 에서는 여러 파일과 디렉토리가 나옵니다

모두 . 으로 시작하는
파일 또는 디렉토리 입니다

. 으로 시작하면
숨김 파일 또는 디렉토리

이 . 은 현재 디렉토리를 의미

find

- -name: 이름으로 필터링
- -type: 디렉토리 또는 파일로 필터링

```
ubuntu@ubuntu-server:~$ find -type f
./.sudo_as_admin_successful
./.bash_history
./.profile
./.bashrc
./.cache/motd.legal-displayed
./.bash_logout
./.ssh/authorized_keys
ubuntu@ubuntu-server:~$ find -type d
.
./.cache
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh -type d
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh -type f
ubuntu@ubuntu-server:~$
```

파일만 보기

디렉토리만 보기

이름이 .ssh 인 디렉토리나 파일 보기

이름이 .ssh 인 디렉토리 보기

이름이 .ssh 인 파일 보기

find

- 탐색 디렉토리를 지정하지 않으면 현재 디렉토리를 기준으로 하위의 모든 파일과 디렉토리의 존재를 보여줍니다
- 탐색 디렉토리를 입력하면 해당 디렉토리를 기준으로 하위에 있는 모든 파일과 디렉토리의 존재를 보여줍니다

- find . 은 현재 디렉토리를 기준으로
- find /tmp 는 /tmp 디렉토리를 기준으로

find

- 아래 두 결과가 같은 이유
- . 이 현재 디렉토리이기 때문

```
ubuntu@ubuntu-server:~$ find . -type f
./.sudo_as_admin_successful
./.bash_history
./.profile
./.bashrc
./.cache/motd.legal-displayed
./.bash_logout
./.ssh/authorized_keys
ubuntu@ubuntu-server:~$ find . -type d
.
./.cache
./.ssh
ubuntu@ubuntu-server:~$ find . -name .ssh
./.ssh
ubuntu@ubuntu-server:~$ find . -name .ssh -type d
./.ssh
ubuntu@ubuntu-server:~$ find . -name .ssh -type f
ubuntu@ubuntu-server:~$
```

```
ubuntu@ubuntu-server:~$ find -type f
./.sudo_as_admin_successful
./.bash_history
./.profile
./.bashrc
./.cache/motd.legal-displayed
./.bash_logout
./.ssh/authorized_keys
ubuntu@ubuntu-server:~$ find -type d
.
./.cache
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh -type d
./.ssh
ubuntu@ubuntu-server:~$ find -name .ssh -type f
ubuntu@ubuntu-server:~$
```

find

- 그러면 루트디렉토리를 탐색해봅시다
- 이것은 마치 MS Windows 에서 C드라이브 전체를 놓고 탐색을 하는 것과 같습니다
- Macbook 에서도 동일한 커맨드를 사용할 수 있습니다

```
ubuntu@ubuntu-server:~$ find /
/
/usr
/usr/games
/usr/lib32
/usr/share
/usr/share/readline
/usr/share/readline/inputrc
/usr/share/locale
/usr/share/locale/bn
/usr/share/locale/bn/LC_MESSAGES
/usr/share/locale/bn/LC_MESSAGES/xdg-user-dirs.mo
/usr/share/locale/bn/LC_MESSAGES/debconf.mo
/usr/share/locale/bn/LC_MESSAGES/update-notifier.mo
/usr/share/locale/fr
/usr/share/locale/fr/LC_MESSAGES
/usr/share/locale/fr/LC_MESSAGES/xdg-user-dirs.mo
```

이 아래로 끝없이 결과가 나옵니다
뭔가 다른 수단이 필요합니다

more

- 우선 화면을 작게 줄여주세요
- more ~/.bashrc
- 매개변수는 파일명입니다
- .bashrc 라는 파일의 내용이 너무 길어서 이를 끊어서 보는 방법

- 그러면 이제 find / 의 결과를 페이지 단위로 끊어서 보는 방법에 대해 생각해봅시다
- find / 의 결과 자체는 파일이 아닙니다
- find / 의 결과 자체는 화면 출력입니다
- 화면 출력을 more에 넘기는 방법에 대해 알아보겠습니다

```
ubuntu@ubuntu-server:~$ more ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

--More--(17%)
```

표준입력(stdin)

- more 에 파일명을 argument로 전달하고 파일의 내용을 입력으로 하여 페이지단위로 보여주었습니다
- 파일명이 아닌 파일의 내용을 직접 전달할 수 있습니다
- more < .bashrc
- more .bashrc 와의 차이
  - more .bashrc: 일단 파일의 경로를 입력받고 그 파일을 열어서 내용을 다시 처리
  - more < .bashrc .bashrc 파일을 more의 표준입력으로 리다이렉트
  - 방향은 왼쪽으로 .bashrc가 more로 향하는 방향

```
ubuntu@ubuntu-server:~$ more .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac
```

이 둘은 결과가 같지만 명령은 다릅니다

```
ubuntu@ubuntu-server:~$ more < .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac
```

표준출력(stdout)

- 표준입력이 있으면 표준출력도 있습니다
- 방향을 바꿉시다
- < 이 아니라 >
- 그러면 출력이 됩니다
- 표준출력을 파일로 리다이렉트

more .bashrc 를 입력했는데 아무것도 안나왔습니다
more .bashrc 의 결과(표준출력) 을 test.txt 라는 파일로 리다이렉트 했기 때문입니다

```
ubuntu@ubuntu-server:~$ more .bashrc > test.txt
ubuntu@ubuntu-server:~$ more test.txt
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac
```

출력되는 결과는 .bashrc 파일이 아니라 test.txt 파일의 내용 입니다
test.txt의 내용이 .bashrc 의 내용으로 기록되어 있습니다

표준에러(stderr)

- 에러메시지는 표준출력으로 출력되지 않습니다
- 출력은 맞는데 표준출력이 아니라 표준에러라고 합니다
- 그러나 표준에러를 표준출력으로 보내는 방법은 있습니다

- 왼쪽에서 오른쪽으로 보낸다(redirect) 라는 의미는 "**>**"
- 표준출력의 의미는 "**1**"
- 표준에러의 의미는 "**2**"
- 표준에러를 표준출력으로 보낸다 **2>&1**
- **2>1 은 표준에러를 1 이라는 파일에 기록하는 것입니다**
- **이와 구분하기 위해 & 를 붙여서 2>&1 로 처리하면 표준에러(2)를 표준출력(1) 로 보냅니다**

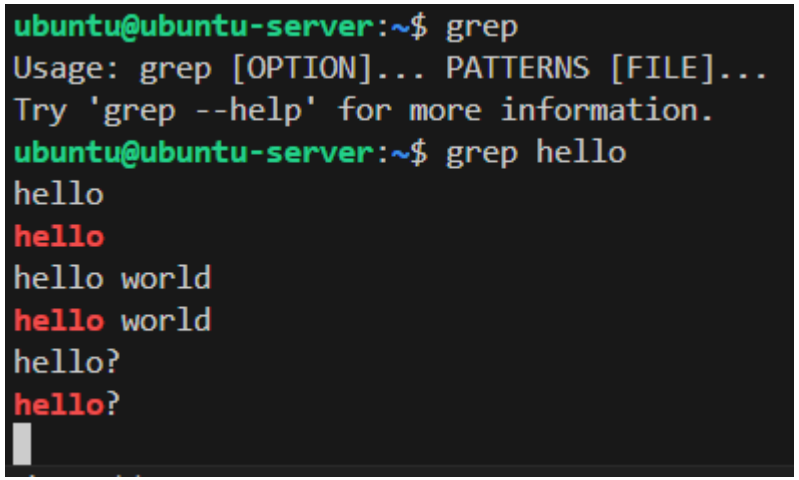
pipe

- pipe: 표준출력을 표준입력 으로 전달하는 방법
- 먼길 돌아서 다시 find / 로 돌아옵니다
- find / 의 결과를 한페이지씩 보고 싶습니다
- find / 의 표준출력을 한페이지씩 보고 싶습니다
- find / 의 표준출력을 more의 입력으로 보내고 싶습니다
- find / 의 표준출력을 more에 pipe 전달하고 싶습니다
- find / | more

```
ubuntu@ubuntu-server:~$ find / | more
/
/usr
/usr/games
/usr/lib32
/usr/share
/usr/share/readline
/usr/share/readline/inputrc
/usr/share/locale
/usr/share/locale/bn
/usr/share/locale/bn/LC_MESSAGES
/usr/share/locale/bn/LC_MESSAGES/xdg-user-dirs.mo
/usr/share/locale/bn/LC_MESSAGES/debconf.mo
/usr/share/locale/bn/LC_MESSAGES/update-notifier.mo
/usr/share/locale/fr
/usr/share/locale/fr/LC_MESSAGES
/usr/share/locale/fr/LC_MESSAGES/xdg-user-dirs.mo
/usr/share/locale/fr/LC_MESSAGES/debconf.mo
/usr/share/locale/fr/LC_MESSAGES/update-notifier.mo
/usr/share/locale/fr/LC_MESSAGES/dpkg.mo
/usr/share/locale/fr/LC_MESSAGES/apt.mo
/usr/share/locale/fr/LC_TIME
/usr/share/locale/fr/LC_TIME/coreutils.mo
/usr/share/locale/bs
/usr/share/locale/bs/LC_MESSAGES
/usr/share/locale/bs/LC_MESSAGES/xdg-user-dirs.mo
/usr/share/locale/bs/LC_MESSAGES/debconf.mo
/usr/share/locale/bs/LC_MESSAGES/update-notifier.mo
/usr/share/locale/bs/LC_MESSAGES/dpkg.mo
/usr/share/locale/bs/LC_MESSAGES/apt.mo
/usr/share/locale/ar
/usr/share/locale/ar/LC_MESSAGES
/usr/share/locale/ar/LC_MESSAGES/xdg-user-dirs.mo
/usr/share/locale/ar/LC_MESSAGES/debconf.mo
/usr/share/locale/ar/LC_MESSAGES/update-notifier.mo
--More--
```

grep

- 페이지 단위로 보면서 우리가 필터링 -> X
- 컴퓨터가 필터링 -> O
- grep만 입력했을 때에는 매개변수를 필수로 요구합니다
- grep 뒤에 지정하는 매개변수는 키보드로부터 입력을 받아서 필터링할 문장을 의미합니다
- grep 뒤에 지정하는 매개변수는 표준입력을 받아서 필터링할 문장을 의미합니다
- grep 뒤에 지정하는 매개변수는 pipe로부터 전달 받아서 필터링할 문장을 의미합니다



grep

- find / | grep /home/ubuntu/.ssh 의 결과가 너무 많이 나옵니다
- permission denied: 권한이 없는 디렉토리에 접근할때 나오는 표준에러

```
ubuntu@ubuntu-server:~$ find / | grep /home/ubuntu/.ssh/
find: ‘/lost+found’: Permission denied
find: ‘/var/cache/pollinate’: Permission denied
find: ‘/var/cache/apt/archives/partial’: Permission denied
find: ‘/var/cache/ldconfig’: Permission denied
find: ‘/var/cache/apparmor/c47eabf7.0’: Permission denied
find: ‘/var/cache/apparmor/e10c1cf9.0’: Permission denied
find: ‘/var/cache/private’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-upower.service-KLPps5’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-systemd-timesyncd.service-rT4ud4’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-systemd-logind.service-1r6RiZ’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-ModemManager.service-LXQh9q’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-fwupd.service-Fri9kD’: Permission denied
find: ‘/var/tmp/systemd-private-83cc9f1838ca4f42ab628a022d813455-systemd-resolved.service-Ja2LLd’: Permission denied
find: ‘/var/lib/update-notifier/package-data-downloads/partial’: Permission denied
find: ‘/var/lib/udisks2’: Permission denied
find: ‘/var/lib/snapd/void’: Permission denied
find: ‘/var/lib/snapd/cookie’: Permission denied
find: ‘/var/lib/apt/lists/partial’: Permission denied
find: ‘/var/lib/private’: Permission denied
find: ‘/var/lib/polkit-1’: Permission denied
find: ‘/var/snap/lxd/common/lxd’: Permission denied
find: ‘/var/spool/rsyslog’: Permission denied
find: ‘/var/spool/cron/crontabs’: Permission denied
find: ‘/var/log/private’: Permission denied
/home/ubuntu/.ssh/authorized_keys
find: ‘/run/udisks2’: Permission denied
```

grep

- 표준에러가 출력되지 않도록 처리합니다
- 표준출력 -> 1
- 표준에러 -> 2
- /dev/null -> null device: 아무것도 아닌 디바이스 = 폐기
- 2>/dev/null -> 2를 /dev/null로 보낸다 -> 표준에러를 /dev/null로 보낸다 ->표준에러를 폐기

```
ubuntu@ubuntu-server:~$ find / 2>/dev/null | grep /home/ubuntu/.ssh
/home/ubuntu/.ssh
/home/ubuntu/.ssh/authorized_keys
ubuntu@ubuntu-server:~$
```

/ 디렉토리 전체를 탐색하는데 permission denied 같은 표준에러는 폐기하고 그 중에서 /home/ubuntu/.ssh 만 필터링

ll이 왜 ls -alF인지 알아봅시다

- more .bashrc | grep alF
- .bashrc 파일에서 alF 라는 내용을 필터링 해봅니다
- alias가 뭔지는 모르지만 ls -alF 에 대한 내용을 .bashrc가 가지고 있는 것은 확실합니다

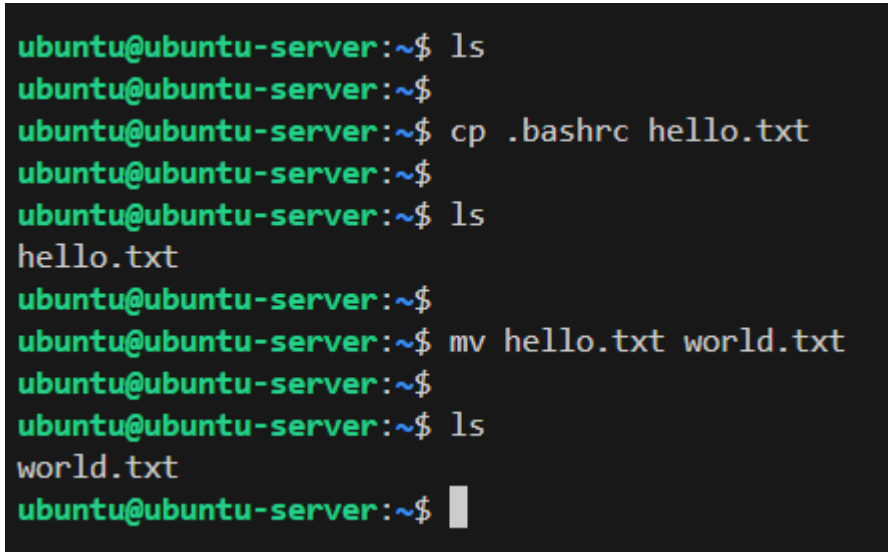
```
ubuntu@ubuntu-server:~$ more .bashrc | grep alF
alias ll='ls -alF'
ubuntu@ubuntu-server:~$
```

ls -alF 를 ll 로 표현하기로 했습니다

파일 복사 및 이동

- 복사: cp (copy)
- ~을 ~ 으로 복사한다
- ~을 ~ 으로 cp 한다
- cp ~을 ~으로

- 이동: mv (move)
- ~을 ~ 으로 이동한다
- ~을 ~ 으로 mv 한다
- mv ~을 ~으로



현재 숨기지 않은 파일은 없습니다

.bashrc 를 hello.txt로 복사합니다

숨기지 않은 파일이 하나 생겼습니다

hello.txt를 world.txt로 이동합니다

숨기지 않은 파일의 이름이 world.txt로 변경되었습니다

중간정리

- 리눅스 터미널을 이용해 아래와 같은 작업이 텍스트 통신만으로 가능함을 확인했습니다

- 현재 작업 디렉토리 확인
- 작업 디렉토리 이동
- 디렉토리내 파일, 디렉토리 리스트 확인
- 디렉토리내 하위 디렉토리, 파일 리스트 확인
- 디렉토리내 하위 디렉토리, 파일 검색 결과 필터링
- 파일, 디렉토리의 복사와 이동

- 따라서 리눅스 터미널을 이용해 MS Window의 탐색기, Macbook의 Finder에 준하는 동작을 할 수 있음을 확인했습니다

패키지 설치 커맨드

- apt 라는 커맨드를 사용합니다
- 패키지가 모여있는 저장소로부터 검색하여 설치하는 방식입니다
- 시스템에 설치하는 커맨드 이므로 관리자 권한이 필요합니다
- 저장소의 정보가 최신화 되어있지 않으면 패키지 검색에 실패하기 때문에 업데이트를 먼저 해주고 설치를 진행합니다
- sudo apt update
- sudo apt install <설치하고자 하는 패키지> 와 같이 설치합니다
- 다음의 패키지를 설치해보겠습니다
- sudo apt install net-tools

현재 시스템의 네트워크 구조 확인하기

- sudo route
- 0.0.0.0 의 의미: 나머지 모든 목적지 = default route = 인터넷 통신 경로

```
ubuntu@ubuntu-server:~$ sudo route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    100    0        0 enp0s8
10.0.3.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s8
_gateway        0.0.0.0         255.255.255.255 UH    100    0        0 enp0s8
kns.kornet.net  _gateway        255.255.255.255 UGH   100    0        0 enp0s8
kns2.kornet.net _gateway        255.255.255.255 UGH   100    0        0 enp0s8
192.168.56.0    0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
ubuntu@ubuntu-server:~$ sudo route -n   -n: IP ->도메인 번역 하지 않기
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.3.2        0.0.0.0         UG    100    0        0 enp0s8
10.0.3.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s8
10.0.3.2        0.0.0.0         255.255.255.255 UH    100    0        0 enp0s8
168.126.63.1    10.0.3.2        255.255.255.255 UGH   100    0        0 enp0s8
168.126.63.2    10.0.3.2        255.255.255.255 UGH   100    0        0 enp0s8
192.168.56.0    0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
ubuntu@ubuntu-server:~$
```

enp0s8: NAT 어댑터
Host 머신과의 통신을 제외한 모든 트래픽의 경로

enp0s3: Host전용 어댑터
오직 Host 머신으로부터 SSH 접속할 때에만 사용

통신 개방 여부 확인하기

- 통신 개방:데이터 수집할때 항상 걸리는 부분
- 통신 개방 여부 확인하는 방법: nc -zv
- 통신 개방 여부만 확인하는 z 옵션을 확인합니다
- verbose 기능을 사용해야 개방 여부를 쉽게 확인할 수 있기 때문에 v 옵션을 사용합니다

- 아래 3개의 커맨드를 실행해보겠습니다
- 이 중에서 success 메시지가 보이지 않는 것은 무엇일까요?
- nc -zv www.google.com 80 (O)
- nc -zv www.google.com 8080 (X)
- nc -zv www.google.com 443 (O)

통신 상태 확인하기

- SSH 연결을 하나 더 열도록 하겠습니다
- 2개의 SSH 연결을 하는 것입니다
- 한쪽에서는 통신이 불가능한 원격지에 통신을 시도합니다
- 다른 한쪽에서는 해당 통신의 상태를 확인합니다
- SYN_SENT: Hello 메시지를 보낸상태(그리고 그 다음 단계가 진행되지 않은 상태)
- 통신 경로상 어떠한 이유에 의해 도달할수 없는 상태일 때 흔히 볼 수 있는 상태 입니다

SSH 연결 1

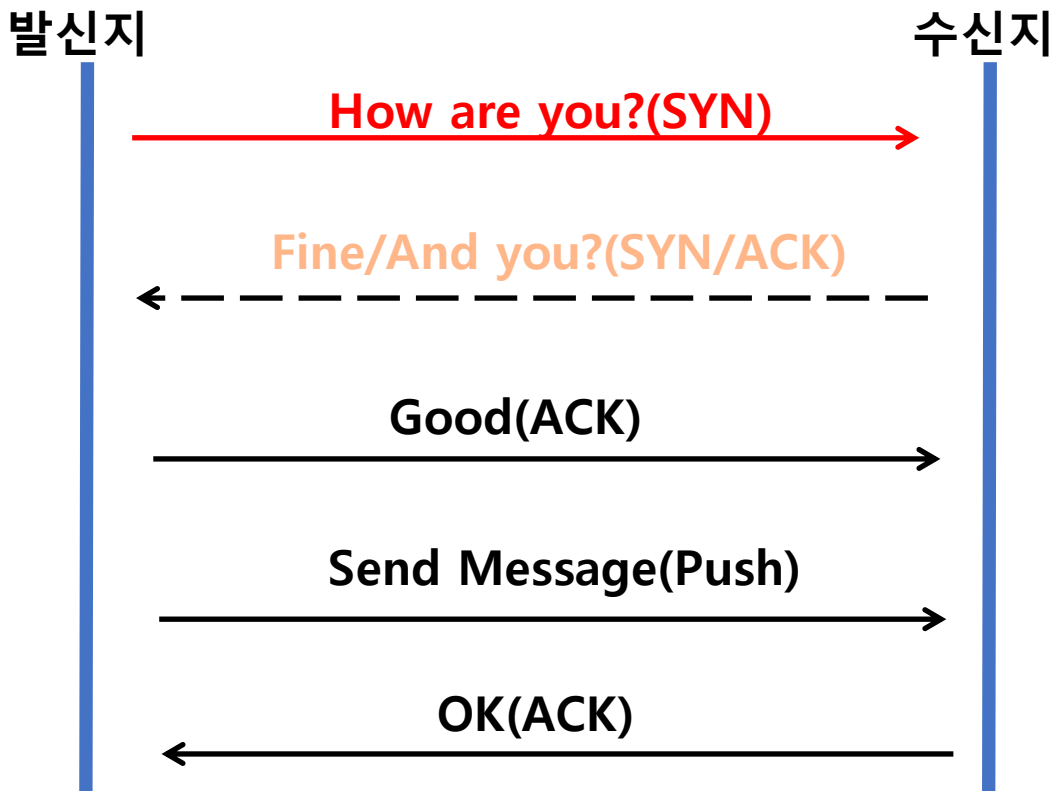
```
ubuntu@ubuntu-server:~$ nc -zv www.google.com 8080
```

SSH 연결 2

```
ubuntu@ubuntu-server:~$ sudo netstat -nap | grep 142.251
tcp        0      1 10.0.3.15:58692         142.251.42.196:8080     SYN_SENT    1978/nc
ubuntu@ubuntu-server:~$
```

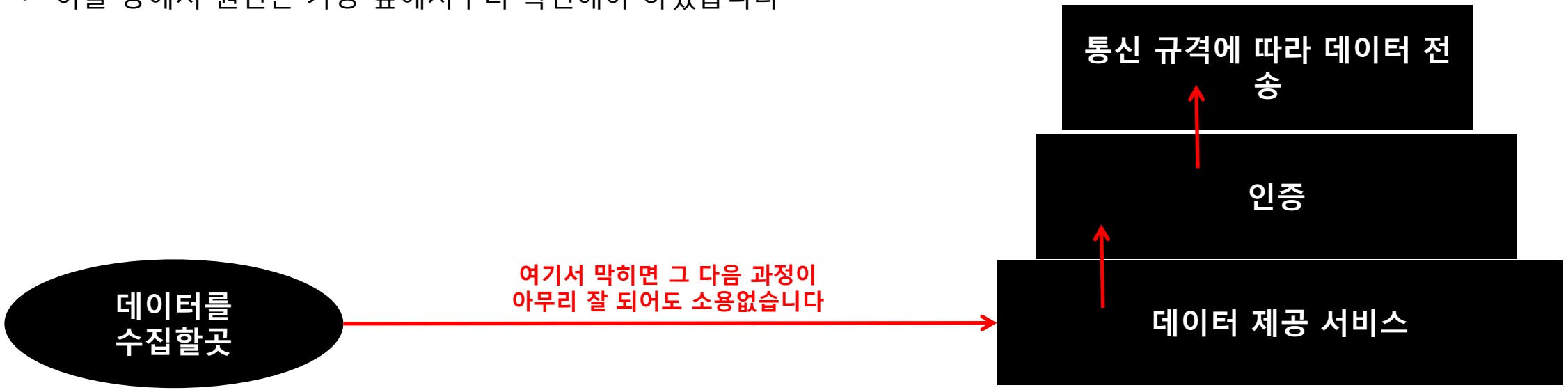
통신 상태 확인하기

- 일반적으로 How are you?(SYN) -> Fine/And you?(SYN/ACK) -> Good(ACK) 와 같은 과정을 거쳐서 통신을 합니다
- 상호간의 발신, 수신 상태를 점검하면서 메시지를 주고받을 준비가 되었음을 확인하는 과정입니다
- SYN_SENT 상태는 How are you? 했는데 응답이 없는 상태가 됩니다



통신 상태 확인이 필요한 이유

- 데이터 수집시 많은 경우 수집 endpoint 및 계정을 제공받게 됩니다
- 따라서 데이터 수집이 되지 않을 경우 크게 원인은 3가지로 나눠볼 수 있습니다하나는 수집 endpoint와의 통신 경로가 개방되지 않았을때
- 또 하나는 계정이 잘못되었을때
- 마지막으로 통신 규격이 맞지 않았을때
- 이들 중에서 원인은 가장 앞에서부터 확인해야 하겠습니다



날씨 데이터 수집하기

- https://open-meteo.com/
- Forecast & Current 탭에 있는 초록색 URL을 클릭해봅시다
- 웹브라우저에 어떠한 데이터가 나오는 것을 확인할 수 있습니다
- 이것을 주기적으로 수집하는 동작을 리눅스 커맨드라인으로 진행해보겠습니다

https://open-meteo.com/ 에서 확인되는 아래 커맨드를 입력해봅니다

```
curl "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"
```

이를 파일로 저장하는 방법은 아래와 같습니다
curl --help
Usage: curl [options...] <url>

```
curl -o weather.json "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"
```

현재 날짜로 파일 이름 만들기

- date --help 를 확인해봅니다
- 상단의 Usage: date [OPTION]... [+FORMAT] 를 참고합니다
- 하단의 $ TZ='Asia/Seoul' date 를 참고합니다
- 아래 커맨드를 입력해봅니다

```
TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.jsonㄹ')  "https://api.open-
meteo.com/v1/forecast?latitude=37.41&longitude=127.00&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"
```

date 커맨드를 이용하여 Timezone이 반영된 date 문자열을 만듭니다

데이터를 수집하여 date 문자열 파일로 저장합니다

이 방법으로 매 수집시마다 파일명을 현재시각으로 하여 저장할 수 있습니다

```
ubuntu@ubuntu-server:~$ TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
20240916-110319_Seoul.json
ubuntu@ubuntu-server:~$ curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json')  "https://
&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5842    0  5842    0     0   5627      0 --:--:--  0:00:01 --:--:--  5633
ubuntu@ubuntu-server:~$ ll 20240916-110328_Seoul.json
-rw-rw-r-- 1 ubuntu ubuntu 5842 Sep 16 02:03 20240916-110328_Seoul.json
ubuntu@ubuntu-server:~$
```

$() 의 의미

- $HOME 에서는 HOME 환경변수의 값을 반환
- $() 안에서는 실행되는 커맨드의 표준출력을 반환(표준에러 제외)
- 둘다 값을 반환한다는 공통점이 있습니다

- 따라서 TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json' 라는 커맨드를 실행해보고 이에 대한 표준출력을 반환받고자 하면 $() 으로 감싸면 됩니다
- echo $(TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json')
- 마치 echo $HOME 하듯이 해봅니다

둘다 뭔가를 echo 했습니다

$HOME을 echo

date 커맨드의 결과를 echo

```
ubuntu@ubuntu-server:~$ echo $HOME
/home/ubuntu
ubuntu@ubuntu-server:~$ echo $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json')
20240916-120240_Seoul.json
ubuntu@ubuntu-server:~$
```

주기적인 수집 - 여러 커맨드를 순차적으로 한줄에

- 10초에 한번씩 데이터를 수집해봅시다
- 반복, 백그라운드 실행(non-blocking 실행), 10초 대기 동작에 대해 알아보겠습니다

- 먼저 아무것도 하지 않고 echo 한줄만 해보겠습니다
- echo “hello world”

- 2개의 커맨드를 한줄로

```
echo "hello world"; echo "hello world2";
```

- 이를 이용해서 10초 단위의 수집 반복동작을 한줄의 커맨드로 실행해보겠습니다

주기적인 수집 - sleep으로 수집주기 설정

- 이어서 아래의 커맨드를 실행해봅니다

```
echo "hello world"; sleep 10; echo "hello world2";
```

- sleep 커맨드의 기능에 대해 확인했습니다
- sleep 10 이라고 하면 우리의 요구사항에 맞겠습니다

- 이제 hello world를 수집 커맨드로 치환해봅니다
- 1회차 수집 ->10초 대기 -> 2회차 수집

```
TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json'
curl -o $(TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json') "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"; sleep 10;
TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json'
curl -o $(TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json') "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m";
```

주기적인 수집 - 반복문에 의한 수집 반복

- 이제 반복문을 사용합니다
- while true; do ~ done 형태로 사용합니다

```
while true; do echo "hello world"; sleep 1; done
```

- ~ 자리를 수집 및 10초대기 커맨드로 치환합니다

```
while true; do TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json'
curl -o $(TZ='Asia/Seoul' date +'%Y%m%d-%H%M%S_Seoul.json') "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"; sleep 10; done
```

주기적인 수집 - 수집 주기 확인

- 10초 주기로 수집이 되고 있는지 확인해봅니다
- 수집자체에 소요되는 시간이 sleep 10에 가산되어 실제로는 10초 이상의 주기로 수집되고 있습니다
- sleep 커맨드를 수집커맨드가 blocking 하는 형태이기 때문입니다

12:51:24
12:51:35
12:51:48
12:51:59
와 같이 정확히 10초 간격으로는
수집되지 않고 있습니다

이 오프셋은 수집자체에
소요되는 시간을 의미합니다

```
ubuntu@ubuntu-server:~$ while true; do TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
> curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json')  "https://api.open-meteo.com
nt=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m"
20240916-125124_Seoul.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5848    0  5848    0     0   4107      0 --:--:--  0:00:01 --:--:--  4109
20240916-125135_Seoul.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5848    0  5848    0     0   2480      0 --:--:--  0:00:02 --:--:--  2480
20240916-125148_Seoul.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5847    0  5847    0     0   5431      0 --:--:--  0:00:01 --:--:--  5434
20240916-125159_Seoul.json
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5847    0  5847    0     0   3372      0 --:--:--  0:00:01 --:--:--  3371
20240916-125211_Seoul.json
```

주기적인 수집 - non-blocking 형태로 수집

- 수집커맨드를 non-blocking 형태로 실행합니다
- 데이터 수집의 완료 여부와 무관하게 10초 sleep 카운트가 시작되는 구조입니다

```
while true; do TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json')  "https://api.open-
meteo.com/v1/forecast?latitude=52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10m" & sleep 10; done
```

10초 단위로 거의 정확하게 수집이 되고 있는 것을 볼 수 있습니다

```
ubuntu@ubuntu-server:~$ while true; do TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
> curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json')  "https://api.open-meteo.c
nt=temperature_2m,wind_speed_10m&hourly=temperature_2m,relative_humidity_2m,wind_speed_10
20240916-125300_Seoul.json
[1] 3130
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5847    0  5847    0     0   5770      0 --:--:--  0:00:01 --:--:--  5777
[1]+  Done                    curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relativ
20240916-125310_Seoul.json
[1] 3135
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5848    0  5848    0     0   6220      0 --:--:-- --:--:-- --:--:--  6214
[1]+  Done                    curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relativ
20240916-125320_Seoul.json
[1] 3140
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5847    0  5847    0     0   5922      0 --:--:-- --:--:-- --:--:--  5918
[1]+  Done                    curl -o $(TZ='Asia/Seoul' date  +'%Y%m%d-%H%M%S_Seoul.json'
52.52&longitude=13.41&current=temperature_2m,wind_speed_10m&hourly=temperature_2m,relativ
20240916-125330_Seoul.json
```

시간, 온도, 풍속만 추출

- 먼저 수집된 파일의 구조를 확인해봅니다
- json 이라는 포맷은 생각하지 않습니다
- 필드 구분자를 comma(,) 로 했을 때 시간, 온도, 풍속은 몇번째 필드가 되는지 확인해봅니다

```
ubuntu@ubuntu-server:~$ awk --help
Usage: awk [POSIX or GNU style options] -f progfile [--] file ...
Usage: awk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options: (standard)
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
```

- 아래 커맨드로 시간, 온도, 풍속을 추출해봅니다
- -F: 필드 구분자(위의 POSIX options 필드로 --field-separator 옵션 입니다)
- {print $12, $14, $15}: 필드 구분자 기준으로 12, 14, 15번째 필드를 출력(위의 'program' 필드입니다)
- *.json: 현재 디렉토리 내의 모든 json 파일(위의 file 필드 입니다)

```
awk -F , '{print $12, $14, $15}' *.json
```

시간, 온도, 풍속만 추출

- 먼저 수집된 파일의 구조를 확인해봅니다
- json 이라는 포맷은 생각하지 않습니다
- 필드 구분자를 comma(,) 로 했을 때 시간, 온도, 풍속은 몇번째 필드가 되는지 확인해봅니다

```
ubuntu@ubuntu-server:~$ awk --help
Usage: awk [POSIX or GNU style options] -f progfile [--] file ...
Usage: awk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options: (standard)
        -f progfile             --file=progfile
        -F fs                   --field-separator=fs
        -v var=val              --assign=var=val
```

- 아래 커맨드로 시간, 온도, 풍속을 추출해봅니다
- -F: 필드 구분자(위의 POSIX options 필드로 --field-separator 옵션 입니다)
- {print $12, $14, $15}: 필드 구분자 기준으로 12, 14, 15번째 필드를 출력(위의 'program' 필드입니다)
- *.json: 현재 디렉토리 내의 모든 json 파일(위의 file 필드 입니다)

```
awk -F , '{print $12, $14, $15}' *.json
```

시간, 온도, 풍속만 추출

- 복수의 구분자를 지정할 수 있습니다
- '[,:{}]' 의 의미: comma, colon, 중괄호, 공백문자를 구분자로 함
- 이를 기준으로 34, 35, 36, 37번째 필드를 출력

```
awk -F  '[,:{} ]'  '{print $34, $35, $36, $37}' *.json
```

```
ubuntu@ubuntu-server:~$ awk -F  '[,:{} ]'  '{print $30, $31, $34, $35, $36, $37}' *.json
"2024-09-16T02 00" "temperature_2m" 13.6 "wind_speed_10m" 16.5
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 30" "temperature_2m" 13.1 "wind_speed_10m" 16.7
"2024-09-16T03 45" "temperature_2m" 13.1 "wind_speed_10m" 16.8
"2024-09-16T03 45" "temperature_2m" 13.1 "wind_speed_10m" 16.8
"2024-09-16T03 45" "temperature_2m" 13.1 "wind_speed_10m" 16.8
```

정렬

- 기온이 가장 높은 시간을 추출해봅니다
- sort: 특정 필드를 기준으로 정렬합니다
- 파일의 내용을 정렬할 수도 있고 표준출력을 pipe로 입력받아 정렬할 수도 있습니다

- 필드의 구분자는 기본 공백문자이며 별도로 지정할 수 있습니다(-t)
- 기본적으로 문자열로 취급하여 정렬하고 숫자로 취급하는 옵션을 지정할 수 있습니다(-n)
- 기본적으로 오름차순 정렬이며 내림차순 정렬 옵션을 지정할 수 있습니다

```
awk -F '[,:{}]' '{print $30, $31, $34, $35, $36, $37}' *.json | sort -n -k 4
```

head (top n)

- 결과가 너무 길어서 우리가 아는 more를 사용해볼 수 있습니다

```
awk -F '[,:{}]' '{print $30, $31, $34, $35, $36, $37}' *.json | sort -n -k 4 | more
```

- 그 외에도 top n 만 확인하기 위해 head를 사용해볼 수 있습니다
- head 커맨드의 옵션을 확인해봅니다 (head --help)

```
awk -F '[,:{}]' '{print $30, $31, $34, $35, $36, $37}' *.json | sort -n -k 4 | head
```

- 반대로 tail도 있습니다

```
awk -F '[,:{}]' '{print $30, $31, $34, $35, $36, $37}' *.json | sort -n -k 4 | tail
```

정리

- 데이터 수집 endpoint에 통신 연결이 되는지 확인하는 방법을 확인했습니다
- 샘플 수집데이터로 날씨 데이터를 수집가능한 사이트를 확인하여 이곳으로부터 데이터 수집해 보았습니다
- 수집된 데이터로부터 우리가 원하는 데이터를 추출하는 방법을 커맨드라인으로 진행해 보았습니다