

# Command-line Environment

2021 Winter ABC Mentoring Program

Team 5 Mentor 정현준

2021 / 01 / 28



# Signal

- 특정 이벤트가 발생하였을 때 신호를 보내서 알려주는 것.
- Interrupt의 한 종류 입니다.
- Ex) Alt+F4 (Window), CTRL + C (LINUX) ...
- kill -l 명령어를 통해 signal 종류에 대해 알 수 있습니다.

```
[cs20151509@uni06 ~]$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

\* Ctrl+C는 SIGINT

\* 각 Signal마다  
고유한 번호가 있다.

# Signal

신호	이식 가능한 번호	기본 동작	설명
SIGABRT	6	종료 (코어 덤프)	프로세스 중단 신호
SIGALRM	14	종료	알람 클럭
SIGBUS	없음	종료 (코어 덤프)	정의되지 않은 메모리 오브젝트의 일부분에 접근.
SIGCHLD	없음	무시	자일드 프로세스 종료, 중단, 계속
SIGCONT	없음	계속	정지하지 않으면 계속 실행.
SIGFPE	없음	종료 (코어 덤프)	오류가 있는 산술 조작.
SIGHUP	1	종료	행업(Hangup).
SIGILL	없음	종료 (코어 덤프)	유효하지 않은 명령.
SIGINT	2	종료	터미널 인터럽트 신호.
SIGKILL	9	종료	킬 (신호를 잡거나 무시할 수 없음).
SIGPIPE	없음	종료	신호를 읽는 사용자가 없는 상태에서 파이프에 기록.
SIGPOLL	없음	종료	폴링 가능한 이벤트.
SIGPROF	없음	종료	프로파일링 타이머 시간 초과.
SIGQUIT	3	종료 (코어 덤프)	터미널 종료 신호.
SIGSEGV	없음	종료 (코어 덤프)	잘못된 메모리 참조.
SIGSTOP	없음	정지	실행 정지 (신호를 잡거나 무시할 수 없음)
SIGSYS	없음	종료 (코어 덤프)	불량 시스템 호출.
SIGTERM	15	종료	종료 신호.
SIGTRAP	없음	종료 (코어 덤프)	트레이스/브레이크포인트 트랩.
SIGTSTP	없음	정지	터미널 정지 신호.
SIGTTIN	없음	정지	백그라운드 프로세스 읽기 시도.
SIGTTOU	없음	정지	백그라운드 프로세스 쓰기 시도.
SIGUSR1	없음	종료	사용자 정의 신호 1.
SIGUSR2	없음	종료	사용자 정의 신호 2.
SIGURG	없음	무시	높은 대역의 데이터를 소켓에서 이용 가능.
SIGVTALRM	없음	종료	가상 타이머 시간 초과.
SIGXCPU	없음	종료 (코어 덤프)	CPU 시간 제한 초과.
SIGXFSZ	없음	종료 (코어 덤프)	파일 크기 제한 초과.

\* POSIX 기준 Signal 표

\* 일부 Signal은 OS에 상관 없이 일관적인 번호를 가집니다.

\* 대부분의 Signal은 특정 상황에서 프로세스를 종료 시키는 역할을 합니다.

\* POSIX = Portable Operating System Interface의 약자  
UNIX를 기반으로 두는 하나의 표준 OS 인터페이스 규칙을 의미합니다.

# Signal


---

- Signal handler가 있으면 특정 signal을 받아도 프로그램을 종료 시키지 않을 수 있습니다.

```
#!/usr/bin/env python
import signal, time

def handler(signum, time):
    print("\nI got a SIGINT, but I am not stopping")

signal.signal(signal.SIGINT, handler)
i = 0
while True:
    time.sleep(.1)
    print("\r{}".format(i), end="")
    i += 1
```



SIGINT를 받게 되면 handler  
함수를 실행시킵니다.

# Signal

---

- Signal handler 적용 전 (signal handler를 주석 처리)

```
[cs20151509@uni06 ABC_program]$ python signal_test.py  
^CTraceback (most recent call last):  
  File "signal_test.py", line 11, in <module>  
    time.sleep(.1)  
KeyboardInterrupt
```



```
[cs20151509@uni06 ABC_program]$ python signal_test.py  
^C  
Catch SIGINT, but not stop  
^C  
Catch SIGINT, but not stop
```

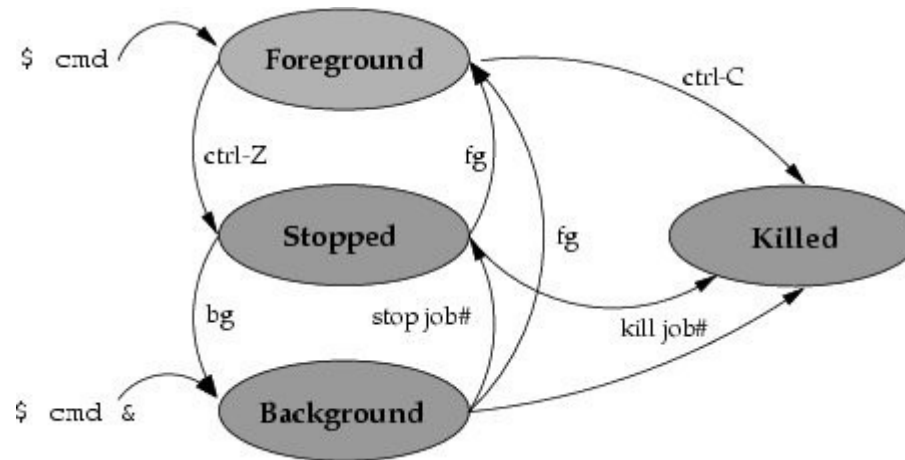
# kill 명령어

---

- Ctrl+C나 Ctrl+W 와 같이 키보드 입력을 통해 signal을 보낼 수 있습니다.
- 그 밖에 terminal을 통해 특정 프로세스에게 signal을 보내는 방법 중에 하나로 kill 명령어를 사용하는 방법이 있습니다.
- **kill -[보내고 싶은 signal] [PID (프로세스 ID)]**
- Ex) kill -9 (또는 -SIGKILL) 13420
  - => SIGKILL signal을 PID 13420인 프로세스에게 보냅니다.
- Grep과 awk를 통해 원하는 프로세스의 ID를 간단하게 검색할 수 있습니다.
- Ex) kill -SIGKILL `ps -ef | grep signal\_test.py | awk '{print \$2}'`

# Foreground & background job

- Ctrl+Z 또는 SIGSTOP signal을 보내면 프로세스는 그 순간부터 실행을 멈추게 됩니다. (suspend 상태라고 합니다.)
- 이는 프로세스가 잠시 멈춘 상태로 완전히 종료된 상태와는 다릅니다.
- 이 상태에서 프로세스를 다시 시작하기 위해서는 foreground로 실행할 것인지, background로 실행할 것인지 선택해야 합니다.



# jobs 명령어

- jobs 명령어를 이용하면 background에 프로세스가 존재하는지 알 수 있습니다.

```
[cs20151509@uni06 ABC_program]$ jobs
[1]+  Running                  python signal_test.py &
[cs20151509@uni06 ABC_program]$ kill -9 %1
[cs20151509@uni06 ABC_program]$ jobs
[1]+  Killed                   python signal_test.py
[cs20151509@uni06 ABC_program]$ jobs
[cs20151509@uni06 ABC_program]$
```

작업의 상태 (Running, Stopped, Killed...)

백그라운드 프로세스 전용 ID (PID와 다릅니다.)



# Background 실행

---

- 명령어나 프로그램 실행에 &를 붙이면 백그라운드 실행이 가능합니다.
- Ex) `python signal_test.py &`
- 이후 `jobs` 명령어를 실행하면 이전 슬라이드처럼 백그라운드 실행중인 프로세스로 나타나게 됩니다.
- Background ID를 알고 있는 경우, %ID를 통해 ID 사용이 가능합니다.
- Ex) `kill -SIGKILL %1`
  - 1번 백그라운드 ID를 가진 프로세스에게 SIGKILL을 보낸다.

# Background 실행

- SIGSTOP signal을 받아 일시정지 된 프로세스를 다시 실행하는 방법은 2가지 입니다.
- 1. fg 명령어를 통해 foreground 실행
- 2. bg 명령어를 통해 background 실행

```
[cs20151509@uni06 ABC_program]$ jobs
[1]-  Stopped                  sleep 1000
[2]+  Stopped                  sleep 2000
[cs20151509@uni06 ABC_program]$ bg %1
[1]- sleep 1000 &
[cs20151509@uni06 ABC_program]$ jobs
[1]-  Running                  sleep 1000 &
[2]+  Stopped                  sleep 2000
[cs20151509@uni06 ABC_program]$ fg %1
sleep 1000
^C
[cs20151509@uni06 ABC_program]$ jobs
[2]+  Stopped                  sleep 2000
```

Background에 2개 정지된 프로세스 존재

bg 명령어로 1번을 background 실행

Background에 2개 프로세스 존재

fg 명령어로 1번을 foreground 실행

Background에 1개 정지된 프로세스 존재

# tmux (Terminal Multiplexer)

---

- 여러 개의 terminal을 사용하고 싶을 때 씁니다.
  - Putty를 여러 개 열면 되면 안되나요...?
- tmux에는 여러 개의 terminal을 열 수 있다는 특징 외에 또다른 특징이 있습니다.
- tmux를 통해 열린 새로운 session에서 실행한 백그라운드 프로세스는 termina을 종료해도 꺼지지 않고 그대로 서버에서 실행됩니다.
- 즉, putty를 종료해도 프로그램이 서버에서 돌아갑니다.

# tmux (Terminal Multiplexer)

---

- `tmux` : 새로운 session을 시작합니다.
- `tmux new -s [이름]` : 새로운 session을 시작한 뒤, session 이름을 정합니다.
- `tmux ls` : 현재 존재하는 모든 tmux session을 출력합니다.
- `tmux a` : 마지막으로 연결했던 session을 불러옵니다.
- `tmux a -t [이름]` : 이름에 해당하는 session과 연결(attach)합니다.
- `Crtl+b` 누른 뒤 `d` : 현재 session과 연결을 해제합니다. (detach)
- `tmux kill-session` : 현재 session을 종료하고 제거합니다.
- `tmux kill-session -t [이름]` : 특정 이름을 가진 session을 제거합니다.

# alias

---

- 자주 쓰는 command가 너무 길 때 사용합니다.
- alias [alias 이름]=[command arg1 arg2 ...] : alias 등록
- unalias [alias 이름] : alias 해제
- Ex) alias gp="ps -ef | grep"

```
[cs20151509@uni06 ABC_program]$ alias gp="ps -ef | grep"
[cs20151509@uni06 ABC_program]$ gp bash
cs20171+   336   1468   0   2020 pts/182   00:00:00 -bash
hspark     529  14043   0   2020 pts/143   00:00:00 -bash
cs20171+  1095   1468   0   2020 pts/93    00:00:00 -bash
cs20171+  1140   1468   0   2020 pts/95    00:00:00 -bash
cs20171+  1469   1468   0   2020 pts/89    00:00:00 -bash
```

# alias

---

- alias만 입력하면 현재 등록된 모든 alias를 볼 수 있습니다.
- 단 alias와 unalias는 현재 session에만 유효합니다.
- 즉, putty를 새로 접속하는 경우 사라지게 됩니다.
- 영구 보존하고 싶은 경우, .bashrc 파일에 저장하면 됩니다.
- 일부 명령어의 경우, 사전에 OS에서 지정한 alias가 존재합니다.
- Ex) mv의 경우, 실제로는 mv -i로 alias가 저장되어 있습니다.

# Dotfile

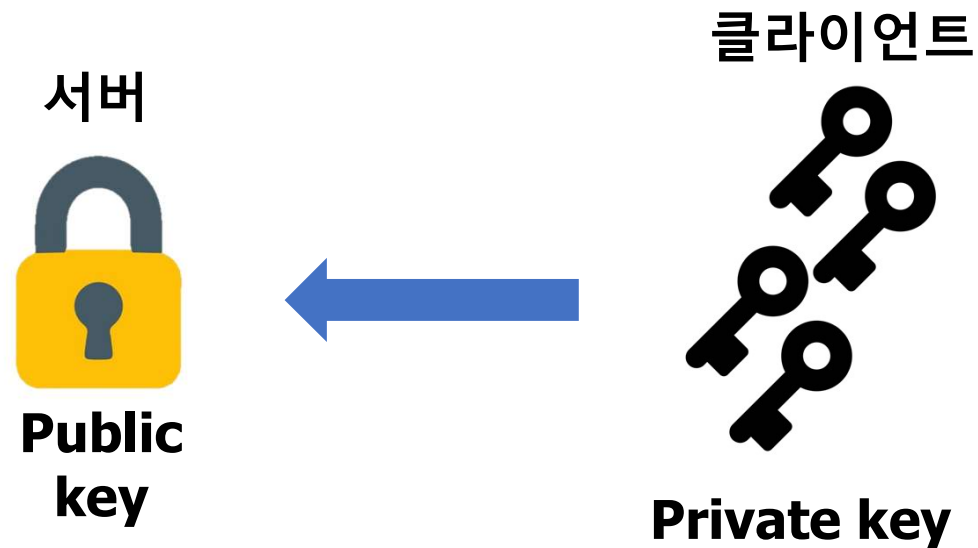
---

- Windows의 숨김 파일 (또는 숨김 폴더) 역할을 합니다.
- ls 명령어로는 보이지 않고 `-a` 같은 옵션을 사용해야 보입니다.
- 주로 어떤 프로그램에 대한 환경 설정(configuration)을 저장할 때 이러한 숨김 파일을 주로 사용합니다.
- Ex) `.bashrc`나 `.vimrc` 혹은 `.gitconfig` 파일 등등...

# SSH key

---

- SSH는 2개의 key를 이용해 서버와 사용자 사이의 접속을 관리합니다.
- 서버 측에서 가진 key를 public key라 합니다.
- 사용자(클라이언트) 측에서 가진 key를 private key라 합니다.





# SSH key

- 서버에 로그인 할 때 SSH key를 미리 등록해두면 나중에 로그인 과정 없이 서버에 접속할 수 있습니다.
- 다만 이러한 key가 유출되면 누구나 서버에 접속할 수 있기 때문에 조심해야 합니다.
- 일반적으로 서버 클러스터를 관리할 때 이러한 과정이 필요합니다.



# SSH key-gen

---

- ssh-keygen 명령어를 입력하면 다음과 같이 key를 저장할 위치와 passphrase를 입력하라고 나옵니다.
- 지금 경우에는 아무것도 입력하지 않고 enter 키를 누른 결과입니다.

```
[cs20151509@uni06 .ssh]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/students_home/old_students/cs20151509/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /students_home/old_students/cs20151509/.ssh/id_rsa.
Your public key has been saved in /students_home/old_students/cs20151509/.ssh/id_rsa.pub.
```

- ~/.ssh 폴더에 id\_rsa와 id\_rsa.pub 파일이 생기면 key가 생성된 것입니다.

```
[cs20151509@uni06 .ssh]$ ls
authorized_keys  id_rsa  id_rsa.pub
```

# SSH key 등록하기

---

- 클라이언트에서 생성된 id\_rsa.pub 파일을 자동 로그인을 원하는 서버의 authorized\_keys 파일에 붙여 넣어야 합니다.
- scp 명령어를 이용해 id\_rsa.pub 파일을 이동시킨 후, 다음과 같은 redirection을 통해 붙여 넣을 수 있습니다.
- `cat ~/id_rsa.pub >> ~/.ssh/authorized_keys`
- 기존의 승인된 key 뒤에 붙여 넣어야 하기 때문에 반드시 `>> redirection`을 사용해주어야 합니다.

# Assignment

---

- 다음 시간은 Debugging과 Profiling 입니다.
- 아래 링크를 미리 보고 오도록 합니다.
- <https://missing.csail.mit.edu/2020/debugging-profiling/>

# Thank you

