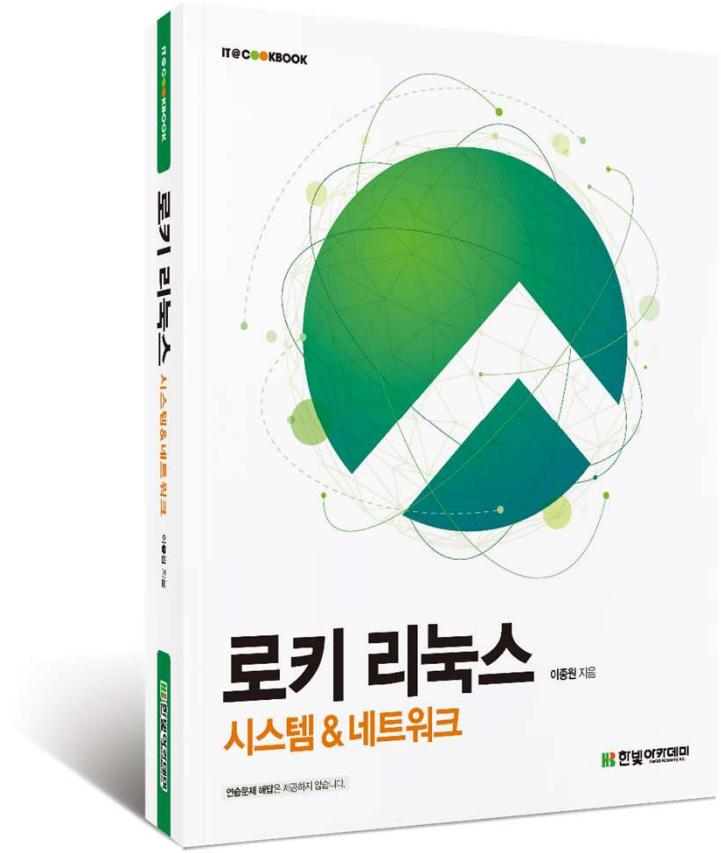

Chapter 06 프로세스 관리

목차

학습목표

- 프로세스가 무엇인지 설명할 수 있다.
- 프로세스의 목록을 확인하고 특정 프로세스를 검색할 수 있다.
- 프로세스를 강제로 종료할 수 있다.
- 프로세스 관리 도구로 전체 프로세스의 상태를 확인할 수 있다.
- 포그라운드와 백그라운드 작업의 차이를 설명할 수 있다.
- 명령을 자동으로 실행하는 방법을 직접 설정할 수 있다.

00 Preview

00 Preview

6장의 내용 구성

- 프로세스는 현재 실행 중인 프로그램을 의미하며, 리눅스에서 사용자가 입력한 명령은 프로세스가 되어 실행됨
- 다양한 프로세스의 종류를 살펴보며, 동작 중인 프로세스를 확인하고 강제로 종료하는 명령을 배움
- 프로세스를 백그라운드에서 동작시키는 방법과 정해진 시간에 자동으로 명령이 실행되도록 설정하는 방법도 배움

01 프로세스의 개념

01 프로세스의 개념

■ 프로세스

- 현재 시스템에서 실행 중인 프로그램을 뜻함

■ 프로세스의 부모-자식 관계

- 필요에 따라 부모 프로세스는 자식 프로세스를 생성하고, 자식 프로세스는 또 다른 자식 프로세스를 만들 수 있음
- 리눅스 시스템을 부팅할 때 스케줄러가 실행한 프로세스인 systemd와 kthreadd 프로세스를 제외하면 모든 프로세스는 부모 프로세스를 가지고 있음
- 자식 프로세스는 부모 프로세스에 의해 만들어지는 프로세스
- 자식 프로세스는 할 일이 끝나면 부모 프로세스에 결과를 돌려주고 종료

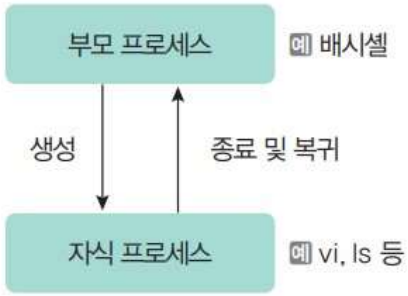

그림 6-1 부모 프로세스와 자식 프로세스의 관계

01 프로세스의 개념

■ 프로세스의 번호

- PID: 각 프로세스는 고유한 번호. 1번부터 시작하고 프로세스가 실행되면서 하나씩 증가하여 부여
- 리눅스가 부팅될 때 PID 1번 systemd 프로세스와 2번 kthreadd 프로세스가 차례로 실행
- 이때 1번 프로세스는 나머지 모든 시스템 프로세스의 부모 프로세스가 되고, 2번 프로세스는 모든 스레드의 부모 프로세스가 됨

■ 프로세스의 부모-자식 관계

- 전통적으로 유닉스에서 1번 프로세스는 init 프로세스
- 로키 리눅스에서 1번 프로세스는 systemd로 init의 역할을 대체
- systemd는 init와 호환성을 유지

01 프로세스의 개념

데몬 프로세스

- 특정 서비스를 제공하기 위해 존재. 평소 대기 상태로 있다가 서비스 요청이 들어오면 서비스를 제공
- 리눅스에서는 다양한 서비스를 제공하기 위한 데몬이 동작함

고아 프로세스

- 자식 프로세스는 종료되면 부모 프로세스로 돌아 감
- 자식 프로세스가 아직 실행 중 인데 부모 프로세스가 먼저 종료되면 자식 프로세스는 고아 프로세스가 됨
- 이 경우 1번 프로세스가 이런 고아 프로세스의 새로운 부모 프로세스가 되어 고아 프로세스가 작업을 마치고 종료될 수 있게 함

01 프로세스의 개념

■ 좀비 프로세스

- 자식 프로세스가 실행을 종료했는데도 프로세스 테이블 목록에 남아 있는 경우 좀비 프로세스라고 함
- 자식 프로세스의 종료 정보를 부모 프로세스가 읽어 가기를 기다리고 있는 것. 부모 프로세스가 자식 프로세스의 종료 정보를 제대로 처리하지 않았기 때문에 이런 일이 발생함
- 좀비 프로세스는 실제로 실행되지는 않지만 동작 중인 프로세스 테이블 목록을 차지하고 있음. 좀비 프로세스가 증가하면 프로세스 테이블의 용량이 부족해서 정상적인 프로세스가 실행되지 않을 수도 있음
- 좀비 프로세스는 kill 명령으로 제거할 수 없으며, SIGCHLD 시그널을 부모 프로세스에 보내어 부모 프로세스가 자식 프로세스를 정리하도록 하거나 부모 프로세스 자체를 종료해야만 함
- 부모 프로세스가 종료되면 좀비 프로세스는 고아 프로세스가 되고, 새로운 부모인 1번 프로세스는 주기적으로 자식 프로세스의 종료 정보를 확인하여 정리

02 프로세스 관리 명령

02 프로세스 관리 명령

■ 프로세스 목록 확인

ps

- **기능** 현재 실행 중인 프로세스에 대한 정보를 출력한다.
- **형식** ps [옵션]
- **옵션** 〈유닉스 옵션〉
  - -e: 시스템에서 실행 중인 모든 프로세스의 정보를 출력한다.
  - -f: 프로세스에 대한 자세한 정보를 출력한다.
  - -u uid: 특정 사용자에 대한 모든 프로세스의 정보를 출력한다.
  - -p pid: pid로 지정한 특정 프로세스의 정보를 출력한다.
  - 〈BSD 옵션〉
  - a: 터미널에서 실행시킨 프로세스의 정보를 출력한다.
  - u: 프로세스 소유자 이름, CPU 사용량, 메모리 사용량 등 상세 정보를 출력한다.
  - x: 시스템에서 실행 중인 모든 프로세스의 정보를 출력한다.
  - 〈GNU 옵션〉
  - --pid PID 목록: 목록으로 지정한 특정 PID 정보를 출력한다.
- **사용 예** ps
  ps -ef
  ps aux

02 프로세스 관리 명령

■ 프로세스 목록 확인

- ps 명령이 출력하는 PID를 통해 프로세스의 부모-자식 관계도 확인할 수 있음
- 로키 리눅스에서 ps 명령의 옵션은 다음과 같은 세 가지 유형을 모두 지원
  - 유닉스(SVR4) 옵션: 묶어서 사용할 수 있고, -으로 시작한다(예. –ef)
  - BSD 옵션: 묶어서 사용할 수 있고, -으로 시작하지 않는다(예. aux)
  - GNU 옵션: - 두 개로 시작한다(예. --pid)
- 세 가지 유형을 섞어서 사용할 수 있으나 이 경우 충돌이 발생할 수도 있음. 여러 가지 옵션을 사용해 보고 자신에게 필요한 정보를 제공하는 옵션 조합을 찾아야 함

02 프로세스 관리 명령

■ 현재 터미널과 관련된 프로세스 목록 출력하기: ps

- 프로세스는 시스템이 직접 실행하거나 특정 터미널에서 실행
- ps 명령을 옵션 없이 사용하면 현재 셸이나 터미널에서 실행한 사용자 프로세스의 정보를 출력

```
[user1@localhost ~]$ ps
    PID TTY          TIME  CMD
  38769 pts/1    00:00:00  bash
  38881 pts/1    00:00:00  ps
```

- PID는 프로세스 번호, TTY는 현재 터미널 번호, TIME은 해당 프로세스가 사용한 CPU 시간의 양
- CMD는 프로세스가 실행 중인 명령이 무엇인지를 알려줌
- bash를 터미널 1번에서 실행 중이고 PID가 38769번임을 알 수 있음
- PID는 시스템에 따라 다르게 나타남

02 프로세스 관리 명령

■ 프로세스 상세 정보 출력하기: -f 옵션

- -f 옵션은 프로세스의 상세한 정보를 출력
- PPID와 C(CPU 사용량), 시작 시간 등의 정보가 추가로 출력

```
[user1@localhost ~]$ ps -f
UID          PID    PPID  C STIME TTY          TIME  CMD
user1      38769   38762  0 16:11 pts/1    00:00:00  -bash
user1      38884   38769  0 16:40 pts/1    00:00:00  ps -f
```

표 6-1 ps -f의 출력 정보

항목	의미	항목	의미
UID	프로세스를 실행한 사용자 ID	STIME	프로세스의 시작 날짜나 시간
PID	프로세스 번호	TTY	프로세스가 실행된 터미널의 종류와 번호
PPID	부모 프로세스 번호	TIME	프로세스 실행 시간
C	CPU 사용량(% 값)	CMD	실행되고 있는 프로그램 이름(명령)

02 프로세스 관리 명령

■ 터미널에서 실행시킨 프로세스 정보 출력하기: a 옵션

- BSD 옵션인 a 옵션도 터미널에서 실행시킨 프로세스의 정보를 출력

```
[user1@localhost ~]$ ps a
    PID  TTY      STAT   TIME  COMMAND
   2072  tty2     Ssl+   0:00  /usr/libexec/gdm-wayland-session --register-session
gnome-session
   2079  tty2     Sl+    0:00  /usr/libexec/gnome-session-binary
   3056  pts/0    Ss+    0:00  bash
  38769  pts/1    Ss     0:00  -bash
  38891  pts/1    R+     0:00  ps a
```

- 콘솔 터미널(tty2)과 가상 터미널(pts/0, pts/1)에서 실행시킨 프로세스 정보를 출력한 것

02 프로세스 관리 명령

■ 터미널에서 실행시킨 프로세스 정보 출력하기: a 옵션

- STAT는 프로세스의 상태를 나타냄

표 6-2 STAT에 사용되는 문자의 의미

문자	의미	비고
R	실행 중(running)	
S	인터럽트가 가능한 대기(sleep) 상태	
T	작업 제어에 의해 정지된(stopped) 상태	
Z	좀비 프로세스(defunct)	
STIME	프로세스의 시작 날짜나 시간	
s	세션 리더 프로세스	BSD 형식
+	포그라운드 프로세스 그룹	
l(소문자 L)	멀티스레드	

02 프로세스 관리 명령

■ 터미널에서 실행시킨 프로세스 상세 정보 출력하기: a 옵션과 u 옵션

- a 옵션과 u 옵션을 함께 사용하면 터미널에서 실행한 프로세스의 상세 정보를 출력함
- a 옵션이나 -f 옵션을 사용한 것과 비교해 보면 CPU와 메모리 사용량 등 추가적인 정보가 출력됨

```
[user1@localhost ~]$ ps au
USER     PID  %CPU  %MEM     VSZ  RSS   TTY    STAT  START TIME  COMMAND
user1   2072   0.0   0.1  374320  3400  tty2   Ssl+  8월20  0:00  /usr/libexec/gdm-
wayland-session --register-session gnome-se
user1   2079   0.0   0.2  513500  4760  tty2   Sl+   8월20  0:00  /usr/libexec/gnome-
session-binary
user1   3056   0.0   0.1  224120  3356  pts/0  Ss+   8월20  0:00  bash
user1  38769   0.0   0.3  224092  5644  pts/1  Ss    16:11  0:00  -bash
user1  38906   0.0   0.2  225480  3688  pts/1  R+    16:46  0:00  ps au
```

표 6-3 ps au의 출력 정보

항목	의미	항목	의미
USER	사용자 계정 이름	VSZ	사용 중인 가상 메모리의 크기(KB)
%CPU	퍼센트로 표시한 CPU 사용량	RSS	사용 중인 물리적 메모리의 크기(KB)
%MEM	퍼센트로 표시한 물리적 메모리 사용량	START	프로세스 시작 시간

02 프로세스 관리 명령

■ 전체 프로세스 목록 출력하기(유닉스 옵션): -e 옵션

- -e 옵션은 시스템에서 실행 중인 모든 프로세스를 출력
- TTY의 값이 ?인 것은 대부분 데몬으로 시스템이 실행한 프로세스
- 출력 결과를 페이지 단위로 확인하려면 |(파이프)와 more나 less 명령을 함께 사용해야 함

```
[user1@localhost ~]$ ps -e | more
    PID TTY          TIME CMD
      1 ?        00:00:13 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
(생략)
     46 ?        00:00:10 kswapd0
     52 ?        00:00:00 kthrotld
--More--
```

- 1번 프로세스가 systemd이고 2번 프로세스가 kthreadd임을 알 수 있음

02 프로세스 관리 명령

■ 전체 프로세스 목록 출력하기(유닉스 옵션): -e 옵션

- 전체 프로세스의 더 자세한 정보를 확인하려면 -e 옵션과 -f 옵션을 함께 사용해야 함
- 두 옵션을 함께 사용할 때는 -ef를 입력함

```
[user1@localhost ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 8월20 ?        00:00:13 /usr/lib/systemd/systemd rhgb
--switched-root --system --deserialize 31
root         2     0  0 8월20 ?        00:00:00 [kthreadd]
root         3     2  0 8월20 ?        00:00:00 [rcu_gp]
(생략)
root        46     2  0 8월20 ?        00:00:10 [kswapd0]
root        52     2  0 8월20 ?        00:00:00 [kthrotld]
--More--
```

- TTY가 ?인 프로세스는 사용자 ID가 root. 스레드는 CMD에 [ ]로 표시하여 구분

02 프로세스 관리 명령

■ 전체 프로세스 목록 출력하기(BSD 옵션): ax 옵션과 aux 옵션

- ax 옵션은 -e 옵션과 마찬가지로 시스템에서 실행 중인 모든 프로세스를 출력

```
[user1@localhost ~]$ ps ax | more
    PID   TTY    STAT    TIME   COMMAND
      1   ?      Ss      0:13   /usr/lib/systemd/systemd rhgb --switched-root --system
--deserialize 31
      2   ?      S       0:00   [kthreadd]
      3   ?      I<      0:00   [rcu_gp]
(생략)
     46   ?      S       0:10   [kswapd0]
     52   ?      I<      0:00   [kthrotld]
--More--
```

02 프로세스 관리 명령

■ 전체 프로세스 목록 출력하기(BSD 옵션): ax 옵션과 aux 옵션

- aux 옵션은 -ef 옵션처럼 시스템에서 실행 중인 모든 프로세스에 대한 자세한 정보를 출력함

```
[user1@localhost ~]$ ps aux | more
USER     PID  %CPU  %MEM     VSZ    RSS   TTY   STAT   START   TIME  COMMAND
root       1   0.0   0.5  172576   9888   ?     Ss     8월20   0:13  /usr/lib/systemd/
systemd rhgb --switched-root --system --deserialize 31
root       2   0.0   0.0       0      0   ?     S      8월20   0:00  [kthreadd]
root       3   0.0   0.0       0      0   ?     I<     8월20   0:00  [rcu_gp]
(생략)
root      45   0.0   0.0       0      0   ?     S      8월20   0:00  [watchdogd]
root      46   0.0   0.0       0      0   ?     S      8월20   0:10  [kswapd0]
--More--
```

02 프로세스 관리 명령

■ 특정 사용자의 프로세스 목록 출력하기: -u 옵션

- -u 옵션을 사용하면 특정 사용자가 실행한 프로세스의 목록을 확인할 수 있음

```
[user1@localhost ~]$ ps -u user1
    PID TTY          TIME   CMD
   2035 ?        00:00:01   systemd
   2042 ?        00:00:00   (sd-pam)
   2058 ?        00:00:00   gnome-keyring-d
   2072 tty2     00:00:00   gdm-wayland-ses
   2074 ?        00:00:00   dbus-broker-lau
(생략)
```

02 프로세스 관리 명령

■ 특정 PID의 프로세스 정보 출력하기: -p 옵션

- -p 옵션과 함께 특정 PID를 지정하면 해당 프로세스의 정보를 출력할 수 있음
- -f 옵션 을 함께 사용하는 것이 좋음

```
[user1@localhost ~]$ ps -p 38769
    PID TTY          TIME     CMD
  38769 pts/1    00:00:00     bash
[user1@localhost ~]$ ps -fp 38769
UID          PID    PPID  C  STIME   TTY        TIME    CMD
user1      38769   38762  0  16:11   pts/1   00:00:00    -bash
```

02 프로세스 관리 명령

■ 특정 프로세스 정보 검색하기: ps와 grep

- ps 명령과 grep 명령을 |로 연결하여 특정 프로세스에 대한 정보를 검색할 수 있음
- 'ps -ef | grep 명령'의 형태로 연결해서 사용
- bash 셸에 대한 정보를 검색한 것

```
[user1@localhost ~]$ ps -ef | grep bash
user1       3056    3007  0  8월20 pts/0    00:00:00 bash
user1      38769   38762  0  16:11 pts/1    00:00:00 -bash
user1      38978   38769  0  17:04 pts/1    00:00:00 grep --color=auto bash
```

02 프로세스 관리 명령

■ 특정 프로세스 정보 검색하기: ps와 grep

- ps 명령과 grep 명령을 |로 연결하여 특정 프로세스에 대한 정보를 검색할 수 있음
- 'ps -ef | grep 명령'의 형태로 연결해서 사용
- bash 셸에 대한 정보를 검색한 것

```
[user1@localhost ~]$ ps -ef | grep bash
user1       3056    3007  0  8월20 pts/0    00:00:00 bash
user1      38769   38762  0  16:11 pts/1    00:00:00 -bash
user1      38978   38769  0  17:04 pts/1    00:00:00 grep --color=auto bash
```

02 프로세스 관리 명령

■ 특정 프로세스 정보 검색하기: pgrep

- pgrep 명령은 ps와 grep을 하나로 통합하여 만든 명령
- 기본적 인자로 지정한 패턴과 일치하는 프로세스를 찾아 PID를 알려주며, 옵션의 지정에 따라 검색 내용이 다양해짐

pgrep

- 기능 지정한 패턴과 일치하는 프로세스의 정보를 출력한다.
- 형식 pgrep [옵션] [패턴]
- 옵션 -x: 패턴과 정확히 일치하는 프로세스의 정보를 출력한다.
  -n: 패턴을 포함하고 있는 가장 최근 프로세스의 정보를 출력한다.
  -u 사용자 이름: 특정 사용자에 대한 모든 프로세스를 출력한다.
  -l: PID와 프로세스 이름을 출력한다.
  -t term: 특정 단말기와 관련된 프로세스의 정보를 출력한다.
- 사용 예 pgrep bash

02 프로세스 관리 명령

■ 특정 프로세스 정보 검색하기: pgrep

- bash 패턴을 지정한 예. bash를 실행하고 있는 프로세스의 PID는 3056과 38769

```
[user1@localhost ~]$ pgrep -x bash
3056
38769
```

- pgrep 명령의 경우 -l 옵션을 지정해도 PID와 명령 이름만 출력함

```
[user1@localhost ~]$ pgrep -l bash
3056 bash
38769 bash
```

02 프로세스 관리 명령

■ 특정 프로세스 정보 검색하기: pgrep

- 더 자세한 정보를 검색하려면 pgrep 명령을 ps 명령과 연결해서 사용 해야함
- pgrep 명령으로 검색하려는 프로세스의 PID를 찾아 ps 명령으로 자세한 정보를 확인하는 것

```
[user1@localhost ~]$ ps -fp $(pgrep -x bash)
UID          PID    PPID  C   STIME   TTY    STAT   TIME   CMD
user1       3056    3007  0   8월20   pts/0  Ss+    0:00   bash
user1      38769   38762  0   16:11   pts/1  Ss     0:00   -bash
```

- -u 옵션으로 사용자명을 지정하면 그 사용자의 프로세스 정보만 검색할 수 있음

```
[user1@localhost ~]$ ps -fp $(pgrep -u user1 bash)
UID          PID    PPID  C   STIME   TTY    STAT   TIME   CMD
user1       3056    3007  0   8월20   pts/0  Ss+    0:00   bash
user1      38769   38762  0   16:11   pts/1  Ss     0:00   -bas
```

02 프로세스 관리 명령

■ 프로세스 종료

- 응답이 없는 프로세스나 불필요한 프로세스를 강제로 종료하려면 해당 프로세스의 PID를 알아야 함
- ps -ef나 ps aux 명령으로 프로세스의 정보를 확인하면 PID와 PPID를 알 수 있음
- 프로세스를 종료하려면 kill이나 pkill 명령을 사용. 이 명령들은 프로세스에 시그널을 보내 프로세스를 종료함
- 시그널: 프로세스에 무언가 발생했음을 알리는 간단한 메시지. 미리 정의된 상수를 사용. 번호로 구분되며 이름을 가지고 있음
- 시그널을 받은 프로세스는 기본적으로 종료
- 리눅스에서 지원하는 시그널의 목록은 kill -l 명령으로 알 수 있음

02 프로세스 관리 명령

■ 프로세스 종료

```
[user1@localhost ~]$ 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
```

02 프로세스 관리 명령

■ 프로세스 종료

표 6-4 주요 시그널

시그널	번호	기본 처리	의미
SIGHUP	1	종료	터미널과의 연결이 끊겼을 때 발생한다.
SIGINT	2	종료	인터럽트로 사용자가 Ctrl+c를 입력하면 발생한다.
SIGQUIT	3	종료, 코어덤프	종료 신호로 사용자가 Ctrl+\를 입력하면 발생한다.
SIGKILL	9	종료	이 시그널을 받은 프로세스는 무시할 수 없으며 강제로 종료된다.
SIGALRM	14	종료	알람에 의해 발생한다.
SIGTERM	15	종료	kill 명령이 보내는 기본 시그널이다.

02 프로세스 관리 명령

■ 프로세스 종료하기: kill

kill

- 기능 지정한 시그널을 프로세스에게 보낸다.
- 형식 kill [-시그널] PID...
- 시그널 2: 인터럽트 시그널을 보낸다(Ctrl+c).
  9: 프로세스를 강제로 종료한다.
  15: 프로세스와 관련된 파일들을 정리하고 종료한다. 종료되지 않는 프로세스가 있을 수 있다.
- 사용 예 kill 1001
  kill -15 1001
  kill -9 1001

- 프로세스는 시그널을 받았을 때 어떻게 처리할 것인지 동작이 지정되어 있음
- 15번 시그널은 일반적으로 프로세스 종료이지만, 시그널을 무시하거나 다른 동작을 하도록 지정되어 있다면 프로세스가 종료되지 않을 수도 있음. kill 명령에서 시그널을 지정하지 않으면 15번 시그널로 간주됨
- 좀비 프로세스의 경우 9번 시그널을 받아도 종료되지 않을 수 있음

02 프로세스 관리 명령

■ 프로세스 종료하기: kill

- 프로세스 종료를 실습하기 위해 터미널을 하나 더 생성하여 두 개의 터미널에서 작업하기
- 터미널 1에서 man ps 명령을 실행하고, 터미널 2에서 해당 프로세스를 검색하여 종료 시킬 것

```
[user1@localhost ~]$ man ps                                    → 터미널 1에서 실행한다.
PS(1)                  User Commands                  PS(1)

NAME
    ps - report a snapshot of the current processes.
(생략)
```

- 터미널 2에서 pgrep과 ps 명령을 실행하여 터미널 1에서 실행한 man 프로세스의 PID를 확인

```
[user1@localhost ~]$ ps -fp $(pgrep -x man)                    → 터미널 2에서 실행한다.
UID          PID    PPID  C STIME TTY          TIME CMD
user1      39418   38769  0 19:23 pts/1    00:00:00 man ps
```

02 프로세스 관리 명령

■ 프로세스 종료하기: kill

- man 프로세스를 종료해 보자. man 프로세스의 PID를 지정하여 kill 명령을 실행함

```
[user1@localhost ~]$ kill 39418
```

- kill 명령을 실행하면 터미널 1에서 man이 종료되어, 터미널 2에서 pgrep 명령으로 검색해도 나오지 않음. man 프로세스가 종료되어 사라진 것

```
(생략)
 Manual page ps(1) line 1 (press h for help or q to quit)종료됨 → 터미널 1의 상태다.
[user1@localhost ~]$
```

```
[user1@localhost ~]$ pgrep -x man                    → 터미널 2에서 실행한다.
[user1@localhost ~]$
```

- 리눅스에서 작업하다 보면 kill 명령만으로 종료되지 않는 경우가 발생함. 이럴 때는 9번 시그널을 사용하여 강제 종료함(kill -9 pid)

02 프로세스 관리 명령

■ 프로세스 종료하기: pkill

- pkill 명령도 kill 명령과 마찬가지로 시그널을 보내지만, PID가 아니라 프로세스의 명령 이름으로 프로세스를 찾아 종료함
- kill 명령과의 차이점은, 명령 이름으로 프로세스를 찾아 종료하므로 같은 명령이 여러 개 검색될 경우 한 번에 모두 종료한다는 것. 자신이 소유한 프로세스만 종료할 수도 있음
- 터미널 1과 터미널 2에서 각각 man pkill을 실행하고 터미널 3에서 프로세스가 동작하는지 확인함

```
[user1@localhost ~]$ ps -fp $(pgrep -x man)  → 터미널 3에서 실행한다.
UID          PID    PPID  C STIME TTY      STAT    TIME    CMD
user1      39517   38769  0 19:31 pts/1    S+      0:00    man ps
user1      39536   39368  0 19:31 pts/2    S+      0:00    man ps
```

```
[user1@localhost ~]$ pkill man              → 터미널 3에서 실행한다.
[user1@localhost ~]$ pgrep -x man
[user1@localhost ~]$
```

02 프로세스 관리 명령

프로세스 종료하기: killall

- killall 명령도 pkill 명령처럼 프로세스의 명령 이름(CMD)으로 프로세스를 찾아 종료함
- 이 이름으로 실행 중인 모든 프로세스를 한 번에 종료할 수 있는데, 물론 해당 프로세스를 소유하고 있어야 함

02 프로세스 관리 명령

■ 프로세스 찾아서 종료시키기

① 터미널 1에서 sh 프로세스를 실행

```
[user1@localhost ~]$ sh
sh-5.1$
```

② 터미널 1에서 more 명령을 실행

```
sh-5.1$ more /etc/services
# /etc/services:
# $Id: services,v 1.49 2017/08/18 12:43:23 ovasik Exp $
(생략)
```

02 프로세스 관리 명령

■ 프로세스 찾아서 종료시키기

③ 터미널 1에서 실행한 프로세스를 터미널 2에서 찾음. -t 옵션이나 a 옵션을 사용함

```
[user1@localhost ~]$ ps -ft pts/1     → 터미널 2에서 실행한다.
UID          PID    PPID  C STIME  TTY        TIME   CMD
user1      38769   38762  0 16:11  pts/1   00:00:00   -bash
user1      39592   38769  0 19:36  pts/1   00:00:00   sh
user1      39595   39592  0 19:37  pts/1   00:00:00   more /etc/services
```

```
[user1@localhost ~]$ ps a
PID TTY        STAT  TIME  COMMAND
2072 tty2      Ssl+  0:00  /usr/libexec/gdm-wayland-session --register-session gnom
2079 tty2      Sl+   0:00  /usr/libexec/gnome-session-binary
3056 pts/0     Ss+   0:00  bash
38769 pts/1    Ss    0:00  -bash
39368 pts/2    Ss    0:00  -bash
39592 pts/1    S     0:00  sh
39595 pts/1    S+    0:00  more /etc/services
39601 pts/2    R+    0:00  ps a
```

02 프로세스 관리 명령

■ 프로세스 찾아서 종료시키기

④ 터미널 1에서 실행한 more 명령을 터미널 2에서 종료함

```
[user1@localhost ~]$ kill 39595     → 터미널 2에서 실행한다.
```

```
ftp-data        20/tcp
--More--(0%)종료됨                  → 터미널 1의 상태다.
sh-5.1$
```

⑤ 터미널 2에서 -9 시그널로 터미널 1에서 실행한 sh 프로세스를 종료함

```
[user1@localhost ~]$ kill -9 39592 → 터미널 2에서 실행한다.
```

```
sh-5.1$ 죽었음                      → 터미널 1의 상태다.
[user1@localhost ~]$
```

02 프로세스 관리 명령

■ 프로세스 찾아서 종료시키기

혼자해보기 프로세스를 찾아서 종료하기

❶ 터미널 1에서 crond 프로세스를 검색한다. 사용자 ID와 터미널 번호가 보이도록 검색한다.

❷ 앞의 ❶번에서 검색한 결과 중 UID가 root인 프로세스를 종료한다. 종료되지 않는다면 그 이유는 무엇인가?

❸ 앞의 ❶번에서 검색한 결과 중 UID가 root인 프로세스를 9번 시그널을 사용하여 종료한다. 종료되지 않는다면 그 이유는 무엇인가?

02 프로세스 관리 명령

top 명령

- top 명령은 현재 실행 중인 프로세스의 정보를 주기적으로 출력하는데, 프로세스의 자세한 요약 정보를 상단에 출력하고 각 프로세스의 정보를 하단에 출력함

표 6-5 top 명령의 출력 정보

항목	의미	항목	의미
PID	프로세스 ID	SHR	프로세스가 사용하는 공유 메모리의 크기
USER	사용자 계정	%CPU	퍼센트로 표시한 CPU 사용량
PR	우선순위	%MEM	퍼센트로 표시한 메모리 사용량
NI	Nice 값	TIME+	CPU 누적 이용 시간
VIRT	프로세스가 사용하는 가상 메모리의 크기	COMMAND	명령 이름
RES	프로세스가 사용하는 메모리의 크기		

02 프로세스 관리 명령

top 명령

- top 명령의 실행 화면

```
top - 19:49:24 up 5 days, 21:37,  4 users,  load average: 0.07, 0.09, 0.14
Tasks: 304 total,   1 running, 303 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   1750.8 total,    146.8 free,   1231.2 used,    536.6 buff/cache
MiB Swap:   2048.0 total,   1378.2 free,    669.8 used.    519.7 avail Mem

    PID USER   PR   NI    VIRT   RES   SHR   S  %CPU   %MEM     TIME+   COMMAND
  39639 user1  20    0  226044  4392  3504   R   5.9    0.2   0:00.01   top
      1 root   20    0  172576  9888  6176   S   0.0    0.6   0:13.78   systemd
      2 root   20    0       0     0     0   S   0.0    0.0   0:00.28   kthreadd
      3 root    0  -20       0     0     0   I   0.0    0.0   0:00.25   rcu_gp
      4 root    0  -20       0     0     0   I   0.0    0.0   0:00.00   rcu_par_gp
(생략)
```

02 프로세스 관리 명령

top 명령

- top 명령 중에는 종료하지 않고 실시간으로 프로세스의 상태를 보여주며 내부적으로 사용하는 명령도 있음

표 6-6 top 명령의 내부 명령

내부 명령	기능
Enter, Back Space	화면을 즉시 다시 출력한다.
h, ?	도움말 화면을 출력한다.
k	프로세스를 종료한다. 종료할 프로세스의 PID를 물어본다.
n	출력하는 프로세스의 개수를 바꾼다.
u	사용자에 따라 정렬하여 출력한다.
M	사용하는 메모리 크기에 따라 정렬하여 출력한다.
p	CPU 사용량에 따라 정렬하여 출력한다.
q	top 명령을 종료한다.

02 프로세스 관리 명령

■ 시스템 정보

- 로키 리눅스의 GUI인 그놈에서 기본으로 제공하는 도구 중에 '시스템 정보'가 있음
- 시스템 정보 화면은 프로세스 이름, 사용자, CPU 사용량, 메모리 사용량 등의 정보를 보여주며 [프로세스 중단(E)] 버튼을 사용하여 프로세스를 종료할 수도 있음
- 시스템 정보에서는 시스템 자원이나 파일 시스템 관련 정보도 확인할 수 있음

그림 6-2 시스템 정보 화면

03 포그라운드 · 백그라운드 프로세스와 작업 제어

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 포그라운드 작업과 백그라운드 작업

- 어떤 명령을 실행하면 그 명령이 끝날 때까지 기다려야 다시 터미널을 사용할 수 있음
- 작업 제어 도구는 한 터미널에서 동시에 여러 프로세스를 실행하고 관리하기 위해 개발되었음

■ 포그라운드 작업

- 포그라운드 프로세스: 사용자가 입력한 명령이 실행되어 결과가 출력될 때까지 기다리는 방식으로 처리되는 프로세스. 작업 제어에서는 이를 포그라운드 작업이라고 함
- 포그라운드 작업은 프롬프트가 다시 출력될 때까지 기다려야 함

```
[user1@localhost ~]$ sleep 100  → 포그라운드 작업
                                → sleep 명령이 끝날 때까지 기다려야 한다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 백그라운드 작업

- 백그라운드 방식으로 명령을 실행하면 명령의 처리가 끝나는 것과 관계없이 곧바로 프롬프트가 출력되어 사용자가 다른 작업을 계속할 수 있음
- 백그라운드 방식으로 처리되는 프로세스를 백그라운드 프로세스라고 하며, 작업 제어에서는 이를 백그라운드 작업이라고 함
- 백그라운드 작업은 명령의 실행 시간이 오래 걸릴 것으로 예상되거나 명령을 실행한 후 다른 작업을 할 필요가 있을 때 많이 사용함
- 명령을 백그라운드로 실행하려면 명령의 마지막에 &(앰퍼샌드) 기호를 추가 함. 백그라운드로 작업을 실행하면 프롬프트가 바로 나옴

```
[user1@localhost ~]$ sleep 100& → 백그라운드 작업
[1] 39871
[user1@localhost ~]$            → 바로 프롬프트가 나와 다른 작업을 할 수 있다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 백그라운드 작업

- 백그라운드 작업의 실행이 끝나면 백그라운드 작업 결과가 터미널 화면으로 그냥 출력되어 현재 작업 화면과 뒤섞일 수 있음
- 백그라운드로 처리할 때는 출력과 오류 방향을 전환하고, 실행 결과와 오류 메시지는 파일로 저장하는 방법을 사용하여 문제를 해결

```
[user1@localhost ~]$ find / -name passwd > pw.dat 2>&1 & → pw.dat에 결과와 오류를 저장한다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 목록 보기: jobs

jobs

- 기능 백그라운드 작업을 모두 보여준다. 특정 작업 번호를 지정하면 해당 작업의 정보만 보여준다.
- 형식 jobs [%작업 번호]
- %작업 번호 %번호: 해당 번호의 작업 정보를 출력한다.
  %+ 또는 %%: 작업 순서가 +인 작업 정보를 출력한다.
  %-: 작업 순서가 -인 작업 정보를 출력한다.
- 사용 예 jobs %1
  jobs

- jobs 명령을 실행한 결과

```
[user1@localhost ~]$ jobs
[1]-  실행중                 sleep 100&
[2]+  나감 1                 find / -name passwd > pw.dat 2>&1
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 목록 보기: jobs

표 6-7 jobs 명령의 출력 정보

항목	출력 예	의미
작업 번호	[1]	작업 번호로서, 백그라운드로 실행할 때마다 순차적으로 증가한다([1], [2], [3],…).
작업 순서	+	작업 순서를 표시한다. • +: 가장 최근에 접근한 작업 • –: + 작업 바로 전에 접근한 작업 • 공백: 그 외의 작업
상태	실행중	작업 상태를 표시한다. • 실행중: 현재 실행 중이다. • 완료: 작업이 정상적으로 종료되었다. • 종료됨: 작업이 비정상적으로 종료되었다. • 멈춤: 작업이 잠시 중단되었다.
명령	sleep 100&	백그라운드로 실행 중인 명령이다.

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 전환하기

표 6-8 작업 전환 명령

명령	기능
Ctrl+z	포그라운드 작업을 중지한다(종료하는 것이 아니라 잠시 중단하는 것이다).
bg %작업 번호	작업 번호가 지시하는 작업을 백그라운드 작업으로 전환한다.
fg %작업 번호	작업 번호가 지시하는 작업을 포그라운드 작업으로 전환한다.

- 현재 포그라운드로 실행 중인 작업을 백그라운드로 전환하려면 우선 Ctrl +z로 작업을 중지해야 함. 그 다음 'bg %작업 번호' 명령으로 작업을 백그라운드로 전환

```
[user1@localhost ~]$ jobs                    → 현재 백그라운드 작업이 없다.
[user1@localhost ~]$ sleep 100               → 포그라운드로 실행한다.
^Z                                           → Ctrl+z로 일시 중지한다.
[1]+  멈춤                  sleep 100        → 일시 중지된 상태다.
[user1@localhost ~]$ bg %1                   → 백그라운드로 전환한다.
[1]+ sleep 100 &
[user1@localhost ~]$ jobs
[1]+  실행중                sleep 100&       → 백그라운드로 실행 중이다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 전환하기

- 백그라운드로 실행 중인 작업을 다시 포그라운드로 전환하려면 'fg %작업 번호' 명령을 사용함
- bg 명령과 마찬가지로 작업 번호를 지정하지 않고 fg 명령만 사용하면 작업 순서가 +인 작업에 적용

```
[user1@localhost ~]$ jobs
[1]+  실행중                 sleep 100&        → 백그라운드로 실행 중이다.
[user1@localhost ~]$ fg                        → 포그라운드로 전환한다.
sleep 100                                      → 포그라운드로 실행 중이다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 종료하기: Ctrl +c

- 포그라운드 작업은 Ctrl +c를 입력하면 대부분 종료됨
- Ctrl +c는 인터럽트 시그널을 포그라운드 프로세스에 전달하며, 인터럽트를 받으면 기본적으로 프로세스를 종료하도록 되어 있음. 프로그램에서 Ctrl +c를 무시하도록 설정한 경우 종료되지 않음
- 포그라운드 작업을 종료하는 또 다른 방법은 다른 터미널에서 해당 프로세스의 PID를 찾아 강제로 종료 하는 것

```
[user1@localhost ~]$ sleep 100  → 포그라운드로 실행 중이다.
^C                              → 강제 종료한다.
[user1@localhost ~]$
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 종료하기: Ctrl +c

- 백그라운드 작업은 kill 명령으로 강제 종료해야 함. 이때 kill 명령의 인자로 PID 대신 '% 작업번호'를 지정해도 됨

```
[user1@localhost ~]$ sleep 100&     → 백그라운드로 실행 중이다.
[1] 39922
[user1@localhost ~]$ kill %1        → 강제 종료한다.
[user1@localhost ~]$ jobs
[1]+  종료됨              sleep 100   → 종료 메시지가 출력된다.
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 로그아웃 후에도 백그라운드 작업 계속 실행하기

- 로그아웃한 다음에도 작업이 완료될 때까지 백그라운드 작업을 실행해야 할 때 nohup 명령을 사용

nohup

- 기능 로그아웃한 후에도 백그라운드 작업을 계속 실행한다.
- 형식 nohup 명령&

- nohup 명령을 사용할 때는 반드시 백그라운드로 실행해야 함
- 별도로 출력 방향 전환을 하지 않으면 명령의 실행 결과와 오류 메시지가 현재 디렉터리에 nohup.out 파일로 자동 저장됨

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 로그아웃 후에도 백그라운드 작업 계속 실행하기

- nohup로 find 명령을 실행했는데 로그아웃한 다음에도 find 명령이 계속 실행되어 결과가 파일로 저장됨

```
[user1@localhost ~]$ nohup find / -name passwd &
[1] 40014
[user1@localhost ~]$ nohup: ignoring input and appending output to 'nohup.out'
exit
```

- 다시 로그인하여 파일 내용을 살펴보면 다음과 같음

```
[user1@localhost ~]$ more nohup.out
find: '/boot/efi/EFI/rocky': 허가 거부
find: '/boot/grub2': 허가 거부
(생략)
/usr/bin/passwd
/usr/share/licenses/passwd
/usr/share/doc/passwd
/usr/share/bash-completion/completions/passwd
(생략)
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 로그아웃 후에도 백그라운드 작업 계속 실행하기

- 명령 실행 시 출력 방향을 전환하면 nohup.out 파일을 생성하지 않고 지정한 파일에 결과와 오류 메시지를 출력함

```
[user1@localhost ~]$ nohup find / -name passwd > pw.dat 2>&1 &
[1] 40017
[user1@localhost ~]$ exit
```

- 다시 로그인하여 파일 내용을 살펴보면 다음과 같음

```
[user1@localhost ~]$ more pw.da
nohup: ignoring input
find: '/boot/efi/EFI/rocky': 허가 거부
find: '/boot/grub2': 허가 거부
find: '/boot/loader/entries': 허가 거부
(생략)
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 관리하기

① 실습 디렉터리를 만들고 이동

```
[user1@localhost ~]$ mkdir Test/ch6
[user1@localhost ~]$ cd Test/ch6
[user1@localhost ch6]$
```

② 백그라운드 작업을 만듦

- 파일을 하나 복사하고 vi 명령을 실행
- Ctrl +z를 실행하여 vi를 일시 정지함

```
[user1@localhost ch6]$ cp /etc/hosts .
[user1@localhost ch6]$ vi hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
~
(생략)
^Z
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 관리하기

② 백그라운드 작업을 만듦

- vi 작업이 일시 정지되면 다음과 같이 '멈춤'으로 표시됨

```
[user1@localhost ch6]$ vi hosts

[1]+  멈춤                 vi hosts
```

③ jobs 명령으로 백그라운드 작업을 확인

```
[user1@localhost ch6]$ jobs
[1]+  멈춤                 vi hosts
```

④ 정지된 작업을 fg 명령을 사용하여 복구

```
[user1@localhost ch6]$ fg
```

03 포그라운드 · 백그라운드 프로세스와 작업 제어

■ 작업 관리하기

⑤ q!로 vi를 종료

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
~
(생략)
:q!
```

혼자해보기 작업 관리하기

❶ sleep 명령을 두 개의 백그라운드로 실행한다.
❷ 현재 실행 중인 백그라운드 작업이나 정지된 작업을 확인한다.
❸ 백그라운드로 실행 중인 sleep 프로세스 중 1번 작업을 종료한다.
❹ 백그라운드 작업을 포그라운드로 전환한다.
❺ 포그라운드 작업을 강제 종료한다.

03 포그라운드 · 백그라운드 프로세스와 작업 제어

vi와 +z

- vi 초보자가 저지르기 쉬운 실수 중 하나가 vi로 작업을 하다가 +z를 눌러 셸로 빠져나오는 것
- 저장 키와 헷갈렸거나 vi를 일시 정지시키고 셸로 나와 작업을 하려고 했을 수도 있음
- 이때 vi를 일시 정지한 것을 깜빡 잊고 다시 vi로 같은 파일을 열면 같은 파일이 vi로 여러 개 열리게 되어 혼란이 생길 수 있으므로 주의해야 함

04 작업 예약

04 작업 예약

■ 정해진 시간에 한 번 실행

at

- **기능** 예약한 명령을 정해진 시간에 실행한다.
- **형식** at [옵션] [시간]
- **옵션** -l: 현재 실행 대기 중인 명령의 전체 목록을 출력한다(atq 명령과 동일).
  -r 작업 번호: 현재 실행 대기 중인 명령 중 해당 작업 번호를 삭제한다(atrm과 동일).
  -m: 출력 결과가 없더라도 작업이 완료되면 사용자에게 메일로 알려준다.
  -f 파일: 표준 입력 대신 실행할 명령을 파일로 지정한다.
- **사용 예** at -m 0730 tomorrow
  at 10:00 pm
  at 8:15 am May 30

04 작업 예약

■ at 명령 설정하기

- at 명령을 사용하여 정해진 시간에 명령을 실행하도록 예약하려면 at 명령 뒤에 시간을 명시함. 그러면 프롬프트가 at>로 바뀌고 작업을 예약할 수 있음

```
[user1@localhost ch6]$ at 09:00 pm
warning: commands will be executed using /bin/sh
at>
```

- at 명령에서 시간을 지정할 때는 'HH:MM' 형태뿐 아니라 'HHMM' 형태도 지원하며 today, tomorrow, hours, days, weeks와 같이 지정할 수도 있음
  - at 4pm + 3 days: 지금부터 3일 후 오후 4시에 작업을 수행
  - at 10am Jul 31: 7월 31일 오전 10시에 작업을 수행
  - at 1am tomorrow: 내일 오전 1시에 작업을 수행
  - at 10:00am today: 오늘 오전 10시에 작업을 수행

04 작업 예약

■ at 명령 설정하기

- at로 실행할 명령은 기본적으로 표준 입력으로 지정함. 명령의 입력을 마치려면 Ctrl +d를 누름. Ctrl +d를 입력하면 화면에는 <EOT>로 출력됨

```
[user1@localhost ch6]$ at 09:00 pm          → 시간을 지정하여 명령을 실행한다.
warning: commands will be executed using /bin/sh
at> /usr/bin/ls -l ~user1 > ~user1/at.out   → 실행할 명령을 지정한다.
at> <EOT>                                   → Ctrl+d를 입력하여 종료한다.
job 1 at Sat Aug 26 21:00:00 2023           → 작업 예약을 완료하였다.
```

- at로 설정한 작업의 결과는 지정된 파일에 저장됨. 만약 표준 출력을 파일로 전환하지 않았다면 작업 결과가 메일로 전달됨. 이 경우 시스템에 메일이 설정되어 있어야 함

04 작업 예약

■ at 작업 파일 확인하기

- at로 생성된 작업 파일은 /var/spool/at 디렉터리에 저장됨.
  이 디렉터리에는 작업 번호로 파일이 생성됨

```
[user1@localhost ch6]$ su -
암호:
[root@localhost ~]# ls -l /var/spool/at
합계 4
-rwx------. 1   user1   user1  3513   8월 26  20:58   a0000101ae90b0
drwx------. 2   root    root      6  10월 26   2022   spool
[root@localhost ~]# exit
[user1@localhost ch6]$
```

- /var/spool/at 디렉터리는 데몬 사용자만 읽을 수 있기 때문에 su 명령을 사용해 root 사용자로 변경한 후 디렉터리의 내용을 확인함

04 작업 예약

■ at 작업 목록 확인하기: -l 옵션과 atq 명령

- at 명령으로 설정된 작업의 목록은 -l 옵션으로 확인할 수 있음. 출력 형식은 각 작업당 한 줄이며 작업 번호, 날짜, 시간, 작업 구분 순

```
[user1@localhost ch6]$ at -l
1       Sat Aug 26 21:00:00 2023 a user1
```

- -l 옵션 외에 atq 명령도 실행 대기 중인 작업의 목록을 출력함

atq

- 기능 현재 사용자의 등록된 작업 목록을 보여준다. 슈퍼유저일 경우 모든 사용자의 작업 목록을 보여준다.
- 형식 atq

```
[user1@localhost ch6]$ atq
1       Sat Aug 26 21:00:00 2023 a user1
```

- 'at -l'과 같은 형식으로 출력됨을 알 수 있음

04 작업 예약

at 작업 삭제하기: -d 옵션과 atrm 명령

- at 명령으로 설정한 작업이 실행되기 전에 삭제하려면 -d 옵션을 사용하고 삭제할 작업 번호를 지정. atrm은 at -d와 같은 기능을 수행함

atrm

- 기능 지정된 작업 번호의 작업을 삭제한다.
- 형식 atrm 작업 번호

04 작업 예약

at 작업 삭제하기: -d 옵션과 atrm 명령

- at 명령으로 작업 두 개를 예약해 보기

```
[user1@localhost ch6]$ at 10:00 am tomorrow
warning: commands will be executed using /bin/sh
at> /usr/bin/ls > ~user1/at1.out
at> <EOT>
job 2 at Sun Aug 27 10:00:00 2023
[user1@localhost ch6]$ at 11:00 pm today
warning: commands will be executed using /bin/sh
at> /usr/bin/ls /tmp > ~user1/at2.out
at> <EOT>
job 3 at Sat Aug 26 23:00:00 2023
```

04 작업 예약

at 작업 삭제하기: -d 옵션과 atrm 명령

- 설정된 작업을 atq 명령으로 확인

```
[user1@localhost ch6]$ atq
2       Sun Aug 27 10:00:00 2023 a user1
3       Sat Aug 26 23:00:00 2023 a user1
```

- 2번과 3번 작업이 예약되어 있음을 알 수 있음
- 작업 번호는 실습에 따라 다를 수 있음. 2번 작업은 at -d 명령으로, 3번 작업은 atrm 명령으로 삭제

```
[user1@localhost ch6]$ at -d 2
[user1@localhost ch6]$ atrm 3
[user1@localhost ch6]$ at -l
[user1@localhost ch6]$
```

04 작업 예약

at 명령 사용 제한하기

- 시스템 관리자는 일반 사용자가 at명령을 사용하도록 허용하거나 사용하지 못하도록 제한할 수 있음. 이와 관련된 파일은 /etc/at.allow와 /etc/at.deny
- at 명령의 사용이 허용된 사용자는 /etc/at.allow 파일에 지정하고, at 명령의 사용이 금지된 사용자는 /etc/at.deny 파일에 지정함
- at.deny 파일은 기본적으로 있지만 at.allow 파일은 없으므로 필요할 때 관리자가 만들어야 함

04 작업 예약

■ at 명령 사용 제한하기

- /etc/at.allow 파일과 /etc/at.deny 파일에는 사용자 이름을 한 줄에 하나씩만 입력함.
  두 파일의 적용 기준은 다음과 같음
  - /etc/at.allow 파일이 있으면 이 파일에 지정된 사용자만 at 명령을 사용할 수 있음. 이 경우 /etc/at.deny 파일은 무시됨
  - /etc/at.allow 파일이 없으면 /etc/at.deny 파일에 지정된 사용자를 제외한 모든 사용 자가 at 명령을 사용할 수 있음
  - 두 파일이 모두 없다면 root만 at 명령을 사용할 수 있음
  - 사용자가 두 파일 모두에 속해 있다면 at 명령을 사용할 수 있음. /etc/at.allow 파일이 적용되기 때문
  - /etc/at.deny를 빈 파일로 두면 모든 사용자가 at 명령을 사용할 수 있는데, 이것이 초기 설정임

04 작업 예약

■ at 명령 사용 제한하기

- /etc/at.allow 파일이 있는데 user1 사용자가 없거나 /etc/at.deny 파일에 user1 사용자가 있다면 at 명령을 실행했을 때 다음과 같이 사용 권한이 없다는 메시지가 출력됨

```
[user1@localhost ch6]$ at
You do not have permission to use at.
```

04 작업 예약

■ 정해진 시간에 반복 실행

- at 명령을 사용하면 미리 정해진 시간에 한 번만 실행함
- 명령을 주기적으로 반복해서 실행해야 한다면 지정 시간이 되면 반복적으로 실행하도록 설정하는 crontab 명령을 사용하면 됨

crontab

- 기능 사용자의 crontab 파일을 관리한다.
- 형식 crontab [-u 사용자ID] [옵션] [파일명]
- 옵션 -e: 사용자의 crontab 파일을 편집한다.
  -l: crontab 파일의 목록을 출력한다.
  -r: crontab 파일을 삭제한다.
- 사용 예 crontab -l
  crontab -u user1 -e
  crontab -r

04 작업 예약

crontab 파일 형식

- crontab 명령으로 관리하는 파일은 사용자별로 생성되는데 이 파일에 반복 실행할 작업이 저장됨
- crontab 파일에는 여러 개의 작업을 저장할 수 있으며 한 행에 하나의 작업을 설정함
- crontab 파일의 한 행은 여섯 항목으로 구성되어 있음

분(0~59)	시(0~23)	일(1~31)	월(1~12)	요일(0~6)	작업 내용

- 앞에서부터 다섯 항목은 시간과 날짜를 나타내는 숫자, 마지막 항목은 반복적으로 수행 할 명령
- 요일은 0이 일요일, 1이 월요일, 6이 토요일을 의미
- 각 항목은 공백 문자로 구분
- 항목의 값이 *이면 해당 항목의 모든 값을 의미

04 작업 예약

■ crontab 파일 형식

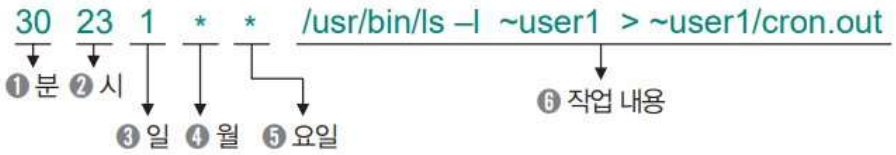

그림 6-3 crontab 파일의 형식

- crontab에 지정하였다면 여기서 ❹번의 월 항목은 매월을, ❺번의 요일은 모든 요일을 의미
- 이 명령은 매월 1일 23시 30분에 지정한 작업을 실행하라는 뜻

04 작업 예약

crontab 파일 생성하고 편집하기: crontab -e

- crontab 파일의 생성과 편집은 crontab -e 명령으로 수행함
- crontab 편집기로는 기본적으로 VISUAL 또는 EDITOR 환경 변수에 지정된 편집기를 사용. 필요할 경우 다음과 같이 명시적으로 편집기를 지정할 수 있음

```
[user1@localhost ch6]$ export EDITOR=vi
```

- crontab -e 명령으로 수행할 작업을 하나 작성함

```
[user1@localhost ch6]$ crontab -e
30 23 1 * * /usr/bin/ls -l ~user1 > ~user1/cron.out
~
(생략)
:wq
```

04 작업 예약

crontab 파일 생성하고 편집하기: crontab -e

- crontab -e 명령으로 편집한 파일을 저장하면 자동으로 /var/spool/cron 디렉터리에 사용자 이름으로 생성됨. 이 디렉터리도 시스템 관리자만 접근할 수 있음

```
[root@localhost ~]# ls /var/spool/cron
user1
```

crontab 파일 내용 확인하기: crontab -l

- crontab -l 명령을 실행하면 현재 사용자의 crontab 파일 내용이 출력됨

```
[user1@localhost ch6]$ crontab -l
30 23 1 * * /usr/bin/ls -l ~user1 > ~user1/cron.out
```

04 작업 예약

crontab 파일 삭제하기: crontab -r

- crontab 파일을 삭제하려면 crontab -r 명령을 사용함
- 만약 시스템 관리자가 특정 사용자의 crontab을 삭제하려면 'crontab -u user1 -r'과 같이 삭제하려는 사용자 ID를 지정함

```
[user1@localhost ch6]$ crontab -r
[user1@localhost ch6]$ crontab -l
no crontab for user1
```

04 작업 예약

crontab 명령 사용 제한하기

- at 명령과 마찬가지로 crontab 명령도 /etc/cron.allow와 /etc/cron.deny 파일로 crontab 명령 사용 권한을 제한할 수 있음. 두 파일에는 사용자 이름을 한 줄에 하나씩만 입력함
- cron.deny 파일은 기본적으로 있지만 cron.allow 파일은 관리자가 만들어야 함
- 두 파일이 적용되는 기준
  - /etc/cron.allow 파일이 있으면 이 파일에 지정된 사용자만 crontab 명령을 사용할 수 있음
  - /etc/cron.allow 파일이 없고 /etc/cron.deny 파일이 있으면 이 파일에 사용자 계정이 없어야 crontab 명령을 사용할 수 있음
  - /etc/cron.allow 파일과 /etc/cron.deny 파일이 모두 없으면 시스템 설정에 따라 시스템 관리자만 crontab 명령을 사용할 수도 있고 모든 사용자가 사용할 수도 있음
  - 만일 두 파일이 모두 있다면 /etc/cron.allow 파일이 적용되고, /etc/cron.deny는 무시됨

04 작업 예약

crontab 명령 사용 제한하기

- 만약 두 파일이 모두 없는데 일반 사용자가 crontab 명령을 사용하려고 하면 다음과 같은 메시지가 출력됨

```
[user1@localhost ch6]$ crontab -e
You (user1) are not allowed to use this program (crontab)
See crontab(1) for more information
```

04 작업 예약

■ crontab 설정하기

① crontab -e 명령으로 crontab 파일을 편집함. 명령의 실행 결과를 확인하기 위해 실습하고 있는 날짜와 시간을 먼저 확인하고 설정함

```
[user1@localhost ch6]$ date
2023. 08. 26. (토) 21:25:44 KST
[user1@localhost ch6]$ crontab -e
30 21 26 * * /usr/bin/ls -l /tmp > /home/user1/Test/ch6/tmp.out
~
(생략)
:wq
```

② crontab -l 명령으로 설정 내용을 확인

```
[user1@localhost ch6]$ crontab -l
30 21 26 * * /usr/bin/ls -l /tmp > /home/user1/Test/ch6/tmp.out
```

04 작업 예약

■ crontab 설정하기

③ crontab에 설정한 시간이 경과한 후 작업 결과를 확인

```
[user1@localhost ch6]$ ls
hosts  tmp.out
[user1@localhost ch6]$ cat tmp.out
합계 0
drwx------. 3 root root 17  7월  9 10:57 systemd-private-8ffa9a5505d343ebad65e618c92bfce2-
ModemManager.service-XTd1Fa
drwx------. 3 root root 17  7월  9 10:57 systemd-private-8ffa9a5505d343ebad65e618c92bfce2-
bluetooth.service-IeKFUN
drwx------. 3 root root 17  7월  9 10:57 systemd-private-8ffa9a5505d343ebad65e618c92bfce2-
chronyd.service-Li1yDE
(생략)
```

04 작업 예약

at, crontab 설정하기

혼자해보기 at, crontab 설정하기

❶ 현재 날짜와 시간을 확인한다.

❷ at 명령으로 현재 시간보다 5분 후에 ls –l/tmp 〉 ~user1/at.tmp를 실행하도록 설정한다.

❸ at 명령으로 설정한 내용을 확인한다.

❹ 5분 후에 at 명령이 실행되었는지 확인한다.

❺ crontab 명령으로 매주 일요일 18시에 user1의 홈 디렉터리 내용을 출력하여 홈 디렉터리 아래에 week.out 파일로 저장하도록 설정한다(실습하는 요일과 시간을 고려하여 결과를 확인할 수 있게 수정한다).

❻ crontab 명령으로 설정된 내용을 확인한다.

❼ crontab 명령으로 설정된 내용의 실행 결과를 확인한다. 어디서 확인할 수 있는가?

Thank you!

한빛아카데미 Hanbit Academy, Inc.