

기본 리눅스 명령어

SPARCS 11 cling



기본 리눅스 명령어

- 로그인
- 파일/디렉토리 관리
- 파일 내용 읽기
- 파일 소유와 권한 변경
- 프로세스, 작업 관리
- Etc...



리눅스 시스템에 로그인하기

- 시스템에 접속하는 여러 사용자(유저)를 계정으로 구분
- 유저에 따라 파일에 대한 권한이 다르다

- login:
- Password:

- 대소문자 구분
- passwd로 비밀번호 설정 및 변경

리눅스 시스템에 로그인하기

- root로 접속했을 경우 / sudo로 루트권한을 얻었을 경우
- #

- 다른 유저로 접속했을 경우
- \$
- Shell Prompt를 나타냄

셸 (Shell)

- 운영체제 내부(커널)와 유저 사이에 명령어 인터페이스를 제공
- 로그인 후 셸 프롬프트가 나온다
 - = "shell level"에 있다
 - = 명령어를 입력할 수 있다.
- echo \$SHELL
 - 현재 사용중인 셸 프로그램의 경로 출력
 - Linux에서는 대부분 bash 사용
- chsh (**change shell**)
 - 패스워드 입력 후 셸 변경 가능

리눅스 명령어 (Command)

- [command] [argument1] [argument2]...
- Command에 따라 사용가능한 argument의 형식 등이 다르다
 - command의 동작 방법을 명시하는 option
 - 파일이나 디렉토리의 이름/경로
 - 임의의 문자열
 - Etc...
- ls -a -l wheel
 - ls는 command이고 -a, -l, wheel은 arguments

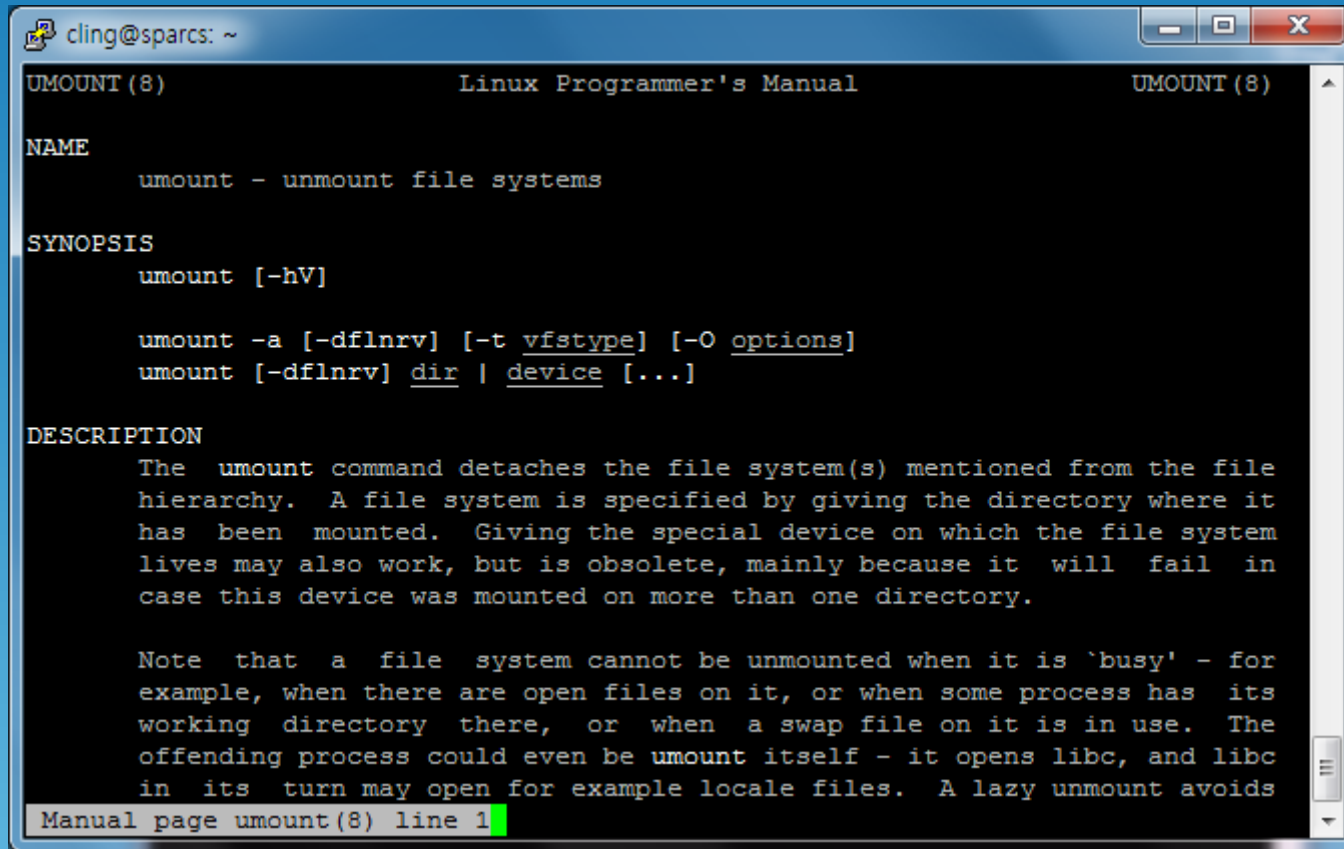
리눅스 명령어 (Command)

- 수많은 명령어를 모두 정확히 기억하기는 어렵다
- man [명령어]
 - **manual**
 - 해당 명령어의 매뉴얼 페이지를 보여준다
- apropos [주제어]
 - = man -k [명령어]
 - ...에 관하여
 - 주제어에 관련된 매뉴얼 페이지의 제목을 찾아준다

리눅스 명령어 (Command)

```
cling@sparcs: ~  
cling@sparcs:~$ apropos mount  
free (1)          - Display amount of free and used memory in the system  
mklost+found (8) - create a lost+found directory on a mounted Linux secon...  
mount (2)         - mount and unmount file systems  
mount (8)         - mount a file system  
mount.nfs (8)     - mount a Network File System  
mountpoint (1)   - see if a directory is a mountpoint  
setup (2)        - setup devices and file systems, mount root file system  
showmount (8)    - show mount information for an NFS server  
sleep (1)        - delay for a specified amount of time  
umount (2)       - mount and unmount file systems  
umount (8)       - unmount file systems  
umount.nfs (8)   - unmount a Network File System  
umount2 (2)      - mount and unmount file systems  
cling@sparcs:~$ man umount
```

리눅스 명령어 (Command)



```
cling@sparcs: ~
Linux Programmer's Manual
UMOUNT (8)
NAME
    umount - unmount file systems
SYNOPSIS
    umount [-hV]

    umount -a [-dflnr] [-t vfstype] [-O options]
    umount [-dflnr] dir | device [...]
DESCRIPTION
    The umount command detaches the file system(s) mentioned from the file hierarchy. A file system is specified by giving the directory where it has been mounted. Giving the special device on which the file system lives may also work, but is obsolete, mainly because it will fail in case this device was mounted on more than one directory.

    Note that a file system cannot be unmounted when it is 'busy' - for example, when there are open files on it, or when some process has its working directory there, or when a swap file on it is in use. The offending process could even be umount itself - it opens libc, and libc in its turn may open for example locale files. A lazy unmount avoids

Manual page umount(8) line 1
```

경로 (Path)

- 파일시스템 내에서의 위치를 나타내는 파일 고유의 이름을 경로 (Path)라고 한다.
- 절대 경로 (absolute path)
 - Working directory와 상관 없이 같은 주소를 나타낸다.
 - 파일시스템의 root(최상위) 디렉토리로 시작한다.
 - /home/user/docs/Letter.txt
- 상대 경로 (relative path)
 - Working directory에 상대적인 위치를 나타낸다.
 - docs/Letter.txt

작업 디렉토리 (Working Directory)

- 각각의 프로세스는 한 디렉토리와 관련되어 있다
 - 한 프로세스가 bulgom.txt의 내용을 boolgom.txt에 복사하려 한다.
 - 이 프로세스의 working directory가 /Wheel이라고 하자.
 - /Wheel/bulgom.txt의 내용을 /Wheel/boolgom.txt에 복사하게 된다.
- 셸의 작업 디렉토리 ≡ 유저의 현재 디렉토리
- 로그인 직후의 작업 디렉토리는 각 유저의 "홈 디렉토리"
- 따로 주소가 주어지지 않는 경우 셸에서 입력되는 명령어는 셸의 작업 디렉토리에서 실행

파일/디렉토리 관리

- ls (**list**): 현재 working directory에 있는 파일 목록을 보여준다
 - ls -a (all): 숨김 파일을 보여준다
 - ls -l (long): 파일의 상세 정보를 보여준다
- mkdir (**make directory**)
 - 현재 디렉토리 안에 새 디렉토리를 생성한다
- rmdir (**remove directory**)
 - 비어있는 디렉토리를 제거한다

파일/디렉토리 관리

```
cling@sparcs: ~/wheel_seminar
cling@sparcs:~/wheel_seminar$ ls
cling.txt
cling@sparcs:~/wheel_seminar$ mkdir a b
cling@sparcs:~/wheel_seminar$ ls
a b cling.txt
cling@sparcs:~/wheel_seminar$ mkdir a/aa/aaa
mkdir: `a/aa/aaa' 디렉토리를 만들 수 없습니다: 그런 파일이나 디렉토리가 없음
cling@sparcs:~/wheel_seminar$ mkdir -p a/aa/aaa
cling@sparcs:~/wheel_seminar$ ls -R
.:
a b cling.txt

./a:
aa

./a/aa:
aaa

./a/aa/aaa:

./b:
cling@sparcs:~/wheel_seminar$ rmdir b
cling@sparcs:~/wheel_seminar$ ls
a cling.txt
```

파일/디렉토리 관리

- mv (**move**): 하나 이상의 파일을 새로운 경로로 옮긴다
 - Wheel은 현재위치의 하위디렉토리
 - mv bulgom boolgom
 - mv bulgom Wheel/bulgom
 - mv bulgom Wheel
 - mv bulgom Wheel/boolgom

파일/디렉토리 관리

- cp (**copy**): 파일을 복사한다
 - cp SourceFile TargetFile
 - cp SourceFile TargetDirectory
 - cp -r SourceDirectory TargetDirectory
- rm (**remove**): 파일을 삭제한다
 - rm -r Directory
 - 디렉토리와 그 안의 모든 파일을 삭제한다

파일 내용 읽기

- cat
 - cat a.txt b.txt
 - a.txt와 b.txt의 내용을 연속해서 보여준다
 - cat -n a.txt
 - 각 행에 번호를 매긴다
- more
 - 내용을 앞에서부터 한 페이지씩 출력한다
 - 엔터를 눌러서 한 줄씩 이동
 - 스페이스바를 눌러서 다음 페이지로 이동

파일 내용 읽기

- less

- more의 이전 페이지로 이동할 수 있는 버전
 - 화살표키, Page Up, Page Down 사용 가능

- more는 파일 내용 전체를 읽어들이고 후 실행되는 반면, less는 현재 페이지만 읽어들이어도 작동한다

- 문자열 검색 기능

- /문자열
- n을 눌러서 다음, N을 눌러서 이전 검색 결과로 이동

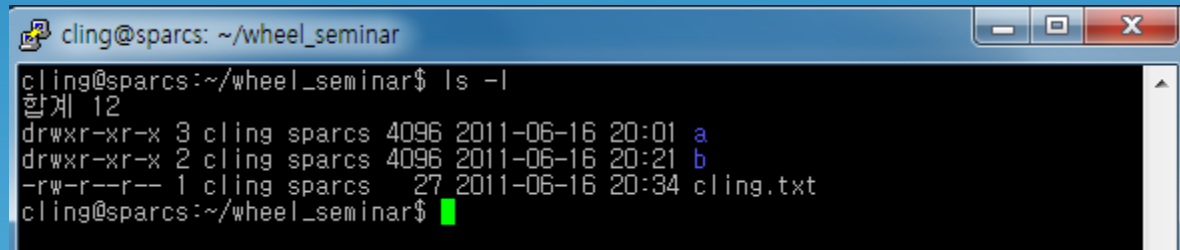
- q를 눌러서 종료

파일 내용 읽기

- wc (**w**ord **c**ount)
 - 파일의 줄 수, 단어 수, 바이트 수를 출력
- grep (**g**lobal **r**egular **e**xpression **p**rint)
 - 파일에서 주어진 정규 표현식(regular expression)에 해당하는 내용이 들어가는 행 출력
 - grep 대한 애국가.txt
 - -v: 해당 문자열이 없는 행만을 출력
 - -w: 해당 문자열을 단어로 포함하고 있는 행을 출력
 - egrep: POSIX extended regular expression을 사용 가능
- vi (vim): 리눅스 기반의 텍스트 에디터

소유(Ownership)와 권한(Permission)

- ls -l
- 파일의 상세 정보(long format)를 보여준다



```
cling@sparcs: ~/wheel_seminar
cling@sparcs:~/wheel_seminar$ ls -l
합계 12
drwxr-xr-x 3 cling sparcs 4096 2011-06-16 20:01 a
drwxr-xr-x 2 cling sparcs 4096 2011-06-16 20:21 b
-rw-r--r-- 1 cling sparcs  27 2011-06-16 20:34 cling.txt
cling@sparcs:~/wheel_seminar$
```

- 파일의 타입, 권한, 링크 수, 소유자, 그룹, 크기, 날짜, 이름

소유(Ownership)와 권한(Permission)

```
drwxr-xr-x 3 cling sparcs 4096 2011-06-16 20:01 a  
drwxr-xr-x 2 cling sparcs 4096 2011-06-16 20:21 b  
-rw-r--r-- 1 cling sparcs 27 2011-06-16 20:34 cling.txt
```

파일 타입

- regular file
- d directory
- b block special file
- c character special file
- l symbolic link
- p pipe
- S domain socket

소유(Ownership)와 권한(Permission)

- 권한: 누군가가 파일을 이용할 수 있는 방법.
 - read: 파일의 내용을 볼 수 있다
 - write: 파일을 바꾸거나 삭제할 수 있다.
 - execute: 파일을 프로그램으로써 실행할 수 있다
 - 일반적으로 파일 생성시 나는 read/write 권한을 가지고 다른이들은 read 권한만을 가진다.
- 디렉토리의 경우
 - read: 디렉토리 안에 있는 파일 이름을 볼 수 있다.
 - write: 디렉토리에 파일을 추가하거나 지울 수 있다
 - execute: 하위디렉토리 내용을 볼 수 있다.
 - 보통 read와 execute는 같이 간다.

소유(Ownership)와 권한(Permission)

- 각 파일/디렉토리는 3가지 레벨의 권한을 가진다
 - Owner/User (소유자)
 - Group (소유 그룹)
 - Other (그 외)
- 각각의 파일/디렉토리에게는 owner와 group이 있다.
- 기본적으로 owner는 파일을 생성한 유저이고 group은 그 유저가 속한 기본 group이다.

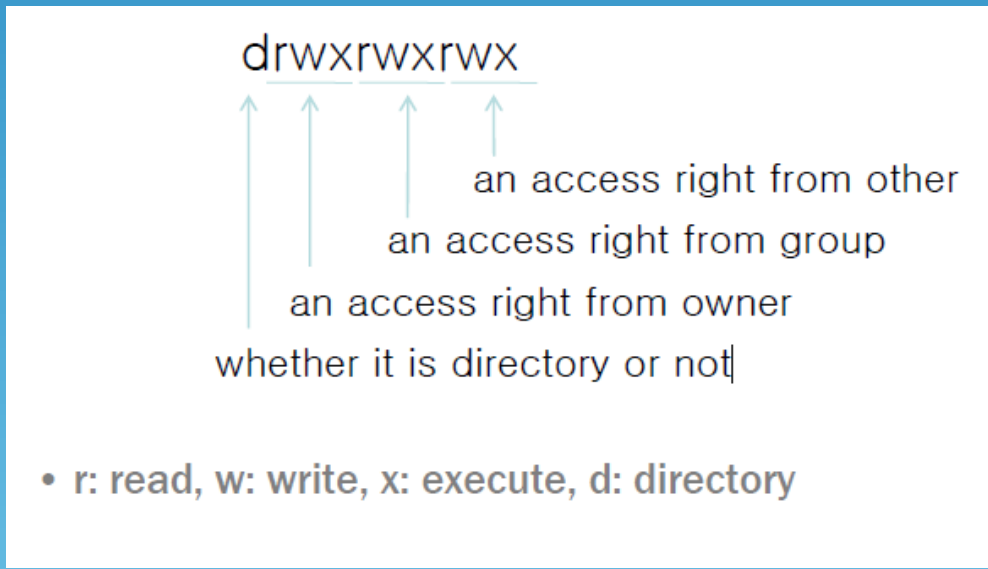
소유(Ownership)와 권한(Permission)

```
drwxr-xr-x 3 cling sparcs 4096 2011-06-16 20:01 a
drwxr-xr-x 2 cling sparcs 4096 2011-06-16 20:21 b
-rw-r--r-- 1 cling sparcs 27 2011-06-16 20:34 cling.txt
```

파일 권한

소유자(owner)

소유그룹(group)



소유(Ownership)와 권한(Permission)

- 대부분의 경우 기본 ownership과 permission으로도 문제가 없다
- 그러나 만일 root 계정으로 다른 유저를 위한 디렉토리를 만들었다면 그 후 **ownership**을 변경해야 할 것이다.
- chown (**change owner**): root만이 owner를 바꿀 수 있다
 - chown [owner] [filename]
 - chown [:group] [filename]
 - chown [owner :group] [filename]
- chgrp (**change group**): 모든 유저가 group을 바꿀 수 있다
 - 단 실행하는 유저가 바꾸는 그룹의 멤버여야 한다
 - chgrp [group] [target1] [target2]...

소유(Ownership)와 권한(Permission)

- **permission의 변경**
- `chmod (u|g|o|a)(+|-)(r|w|x) files directory...`
 - **change mode**
 - user(owner), group, others, all
 - +는 권한 추가, -는 권한 삭제
 - r은 read, w는 write, x는 execute
- `chmod ug+rw mydir`
 - mydir의 owner와 group에게 read와 write권한을 부여한다.
- `chmod a-w myfile`
 - myfile의 owner, group, other에게서 write 권한을 제거한다.

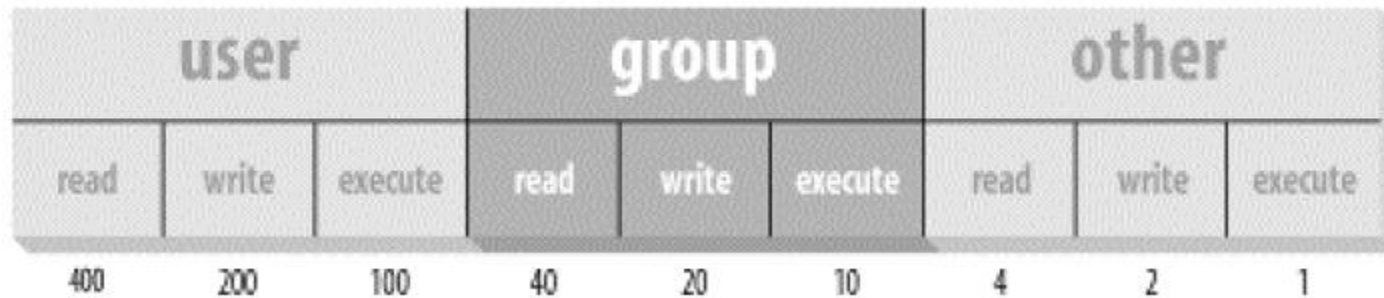
소유(Ownership)와 권한(Permission)

- `umask` (**u**ser **m**ask)
- 파일/디렉토리 생성시 부여되는 기본 권한을 설정
- 단, 파일은 기본적으로 실행 권한을 가지지 않는다
 - `$ umask u=rwx,g=rwx,o=`
 - `$ mkdir fu`
 - `$ vi bar`
 - `$ ls -l`
 - `drwxrwx--- 2 cling cling 512 Sep 1 20:59 fu`
 - `-rw-rw---- 1 cling cling 0 Sep 1 20:59 bar`

소유(Ownership)와 권한(Permission)

- Octal notation (=absolute mode, 8진수 모드)
 - 각 클래스의 permission을 나타내는 rwx 3글자를 2진수 3자리로 된 한 숫자로 해석하여 8진수로 나타내는 것
 - `chmod 664 myfile` -> `rw_rw_r__`
 - 단 `umask` 명령어에는 원하는 권한의 8의 보수를 쓴다

Figure 4-3. Bits in absolute mode



소유(Ownership)와 권한(Permission)

- root만이 읽고 쓸 수 있는 파일이나 디렉토리를 일반 사용자에게 권한을 부여하는 등의 특별한 퍼미션이 있다
 - 심볼릭 모드에서는 rwx외의 글자로 표현
 - 8진수 모드에서는 MSD(가장 좌측수)로 표현
- setuid, setgid, sticky bit
 - setuid: 이 퍼미션이 설정된 파일은 실행되는 동안 owner 권한을 가진다
 - setgid: 이 퍼미션이 설정된 파일은 실행되는 동안 group 권한을 가진다
 - sticky bit: 이 퍼미션이 설정된 디렉토리 안에 있는 파일은 owner와 root만이 삭제할 수 있다. 일반 파일에서는 무시된다.

소유(Ownership)와 권한(Permission)

- setuid (**set user ID** upon execution)
 - Symbolic: owner의 x 필드(실행 필드)에 's'로 표현
 - Octal: 4000으로 표현
- setgid (set group ID upon execution)
 - Symbolic: group의 x필드(실행 필드)에 's'로 표현
 - Octal: 2000으로 표현
- sticky bit
 - Symbolic: other의 x필드(실행 필드)에 't'로 표현
 - Octal: 1000으로 표현

소유(Ownership)와 권한(Permission)

- In (**link**): 링크를 생성한다
- Symbolic (soft) link: 어떤 다른 파일의 경로를 가리키는 파일
 - In -s test.txt hello.txt
 - cat hello.txt
 - test.txt를 삭제하는 경우 hello.txt는 무용지물이 된다
- Hard link: 파일의 실제 물리적 위치를 가리킨다. 한 파일에 여러 이름이 있다고 생각하면 된다.
 - In test.txt hello.txt
 - test.txt를 삭제하더라도 hello.txt 파일은 내용을 간직하고 있다.

작업 관리 (Job Control)

- 리눅스에서 명령어(command)는 파일이다.
 - ls는 bin에 저장된 파일이고, 절대경로인 /bin/ls를 써서 실행할 수 있다
 - 새로운 유틸리티를 제공하려면 명령어가 저장되는 기본 디렉토리에 설치
 - 그러나 파일이 기본 디렉토리에 없다면 명령어를 찾을 수 없다는 메시지가 출력된다.
- echo \$PATH
 - 셸이 명령어를 검색하는 기본 폴더를 출력한다
 - 어디서 많이 본 것 같은데...?
- echo \$SHELL 을 입력하여 사용중인 셸의 경로를 출력했었다.

환경 변수 (Environment Variable)

- \$SHELL, \$PATH = 셸의 환경 변수
- 환경 변수 (environment variable)
 - 각 프로세스가 동작하는 방법에 영향을 끼치는 값들
 - \$SHELL: 셸 프로그램의 위치
 - \$PATH: 셸이 명령어를 검색하는 기본 경로
 - 문자열을 출력하는 명령어인 echo를 이용하여 확인할 수 있다
- export [변수명=값]
 - 환경 변수의 값을 설정하는 명령어
 - export PATH=\$PATH:/user/sbin
 - 기존의 PATH에 /user/sbin을 추가한다

프로세스 (Process)

- 프로세스: 컴퓨터 내에서 독립적으로 실행되고 있는 각각의 프로그램
 - 메모리와 디스크 같은 자원은 한정되어 있기에 커널이라는 강력한 프로그램에서 관리한다.
 - 다른 모든 프로그램은 프로세스이다.
 - 명령어가 입력되면, 셸이라는 프로세스가 새로운 프로세스를 생성한다. (forking)
 - 셸과 명령어 프로그램이 독립된 프로세스이기에 셸은 화면에, 명령어의 결과는 파일에 출력하도록 할 수 있다. (Redirection)
 - 각각의 프로세스는 Process ID(pid)로 구분한다.

프로세스 (Process)

- 셸은 프로세스를 foreground와 background로 나누어 관리한다.

- **Foreground**

- 유저에게 출력되고 입력을 받는 프로세스
- 명령어는 기본적으로 foreground로 실행된 후 종료 / background로 전환된다
- Foreground가 끝난 후에야 새로운 명령어를 받는 셸 프롬프트가 출력된다.

- **Background**

- 유저에게 보여지지 않는다
- Foreground 프로세스가 없는 사이사이에 “뒤에서” 실행된다.
- 명령어 뒤에 &를 붙여서 입력하면 background로 실행된다.

- 이와 같이 프로세스를 관리하는 것을 작업 관리(Job Control), 관리되는 프로세스를 작업(Job)이라고 한다.

작업 관리 (Job Control)

- 각각의 작업을 구분하는 명칭을 jobspec (job specification)이라고 하며 %를 앞에 붙여서 표시한다.
 - 가령 작업번호가 1인 작업은 %1로 나타낼 수 있다.
- Foreground job 실행 중에 Ctrl+Z를 누르면 일시정지(suspend)되고 셸로 컨트롤이 넘어간다.
 - bg [jobspec] – 정지된 작업을 background로 재시작한다.
 - fg [jobspec] – 작업을 현재 foreground job으로 재시작한다.
- jobs [-lnprs]
 - 현재 셸에 있는 작업들의 상태를 보여준다
 - -l 프로세스 ID를 추가로 보여준다
 - -r 실행중인 작업만을 보여준다 (running)
 - -s 정지된 작업만을 보여준다 (stopped)
 - jobs [jobspec]: 해당하는 작업의 정보를 보여준다.

프로세스 (Process)

- 실행중인 프로세스 확인하기
 - ps

```
$ ps
  PID TTY          STAT TIME   COMMAND
 1663 pp3        S      0:01   -bash
 1672 pp3        T      0:07   emacs
 1676 pp3        R      0:00   ps
```

PID - Process ID (used to kill a process) **TIME** - CPU time used so far
TTY - Controlling terminal **COMMAND** - Command running
STAT - State

- PID: 프로세스 ID(번호)
- TTY: 입출력 터미널 (background 데몬은 터미널이 없다)
- STAT: 프로세스의 상태 (Sleeping / sTopped / Running)
- TIME: 프로세스가 사용한 CPU 시간
- COMMAND: 프로세스를 실행한 명령어

프로세스 (Process)

- top

- 실행중인 프로세스의 자세한 정보를 출력하되, 내용이 주기적으로 갱신
- <http://jace.tistory.com/96>

```
bsh@vc201.kaist.ac.kr: ~
top - 20:46:30 up 4 days, 3:56, 2 users, load average: 0.00, 0.00, 0.00
Tasks: 32 total, 1 running, 29 sleeping, 1 stopped, 1 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
Mem: 8388608k total, 139304k used, 8249304k free, 0k buffers
Swap: 0k total, 0k used, 0k free, 0k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	10380	828	696	S	0	0.0	0:00.73	init
514	syslog	20	0	12296	780	592	S	0	0.0	0:00.09	syslogd
537	messageb	20	0	21880	1676	740	S	0	0.0	0:00.29	dbus-daemon
551	root	20	0	40112	1864	1540	S	0	0.0	0:00.02	NetworkManager
563	root	20	0	24128	1260	1044	S	0	0.0	0:00.00	NetworkManagerD
574	root	20	0	35192	1216	948	S	0	0.0	0:00.00	system-tools-ba
597	bind	20	0	129m	16m	1860	S	0	0.2	0:00.18	named
626	root	20	0	50916	1208	724	S	0	0.0	0:00.02	sshd
670	root	20	0	19332	936	744	S	0	0.0	0:00.00	xinetd
712	root	20	0	66424	2032	592	S	0	0.0	0:03.96	sendmail-mta
762	root	20	0	23364	1408	1004	S	0	0.0	0:01.56	ntpd
803	root	20	0	6272	840	712	S	0	0.0	0:02.61	dhcdbd
828	root	20	0	113m	2128	1228	S	0	0.0	0:00.00	gdm
845	root	20	0	18616	972	756	S	0	0.0	0:00.05	cron
864	root	20	0	108m	2720	1204	S	0	0.0	0:01.12	apache2
865	www-data	20	0	108m	2656	1068	S	0	0.0	0:00.00	apache2
880	nobody	20	0	49300	2012	1060	S	0	0.0	0:26.79	gmond

시그널 (Signal)

- Signal(신호)
 - 프로세스간에 의사소통을 위해서 보내는 '신호'
 - ID(번호)나 Name(이름)으로 나타낼 수 있다.
 - Ctrl+Z를 누르면 프로세스에 SIGTSTP이라는 신호가 가서 일시정지된다.

ID	Name	Default Action	Corresponding Event
2	SIGINT	Terminate	Interrupt from keyboard (<code>ctrl-c</code>)
9	SIGKILL	Terminate	Kill program (cannot override or ignore)
11	SIGSEGV	Terminate & Dump	Segmentation violation
14	SIGALRM	Terminate	Timer signal
17	SIGCHLD	Ignore	Child stopped or terminated
20	SIGTSTP	Stop until next SIGCONT	Stop signal from terminal

시그널 (Signal)

- 프로세스에 시그널 보내기
 - kill계열 명령어 (kill, killall, skill...)
 - 시그널이 명시되지 않으면 15번 SIGTERM(terminate)가 가서 프로세스를 죽이므로 kill.
- kill [-signal name / -sigid] [jobspec / pid]
 - 작업 혹은 프로세스에 시그널을 보낸다.
 - kill 1234
 - kill -TERM 1234 kill -KILL 1234
 - kill -15 1234 kill -9 1234
- killall [-signal name / -sigid] [command name]
 - 지정된 명령이 실행중인 모든 프로세스에 시그널을 보낸다
 - killall -KILL httpd
 - 웹 서버 데몬(httpd)를 모두 종료
- skill [-signal name / sigid] [username]
 - 프로세스의 유저에 따라 시그널을 보낸다

시그널 (Signal)

- 시스템 사용을 마치고 로그아웃할 때는 sigid 1번의 HUP(hang up) 시그널이 시스템으로 전달된다
- 이 시그널은 쉘 상태에서 실행중인 모든 프로세스를 종료시키고 로그아웃하게 한다.
- nohup [command] &
 - 이렇게 실행된 명령어는 SIGHUP을 무시한다
 - 이처럼 사용자에게 직접적으로 제어받지 않고 로그아웃 후에도 background에서 계속 실행되면서 여러 작업을 하는 프로그램을 Daemon이라고 한다.
 - 대부분의 데몬은 부팅 시에 시스템에 의해 실행된다
 - httpd: 웹 서버에서 클라이언트나 사용자의 요구를 기다린다
 - udevd: 하드웨어 환경을 설정하는 장치관리자
 - crond: 주기적인 작업을 실행하는 스케줄러

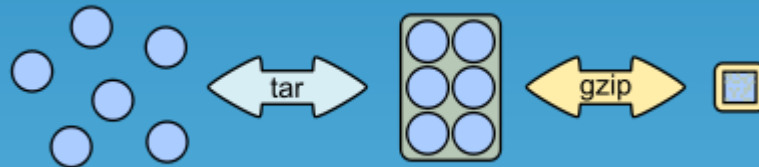
시스템 종료

- shutdown
 - 시스템을 정지시키는 명령어
 - 접속해 있는 사용자에게 시스템이 종료된다는 메시지를 보낸다
 - 새로운 사용자의 로그인을 금지한다
 - 지정된 시간내에 종료되지 않은 프로세스, 사용자를 강제종료한다
 - 메모리에 남아있는 데이터를 디스크에 저장한다
 - mount 되어있는 장치를 unmount한다
 - 시스템을 종료한다
- reboot
 - 시스템 종료 후 재부팅시킨다
- wall [메시지]
 - 접속한 모든 유저에게 메시지를 보낸다
 - root에서 종료 전에 shutdown 메시지를 보내는 데도 쓰인다

tar와 gzip을 이용하여 압축하기

- tar

- 압축하기 전 여러 파일을 하나의 파일로 합치는 명령어
- 실제로 압축을 하기 위해서는 gz, zip, bz2 등이 사용된다
- 결과적으로 .tar.gz 같은 확장자의 압축 파일이 된다



tar와 gzip을 이용하여 압축하기

- tar [cxvzf] filename
 - c (create): 압축하기
 - x (extract): 압축풀기
 - v (verbose): 작업 진행상황을 화면에 출력
 - z (gzip): gzip으로 압축 또는 해제
 - f (file): 파일로 저장한다
- tar.gz 압축하기
 - tar cvf filename.tar file1...
 - gzip filename.tar
- tar.gz 한번에 압축하기
 - tar cvzf filename.tar file1...
- tar.gz 압축풀기
 - gunzip filename.tar.gz
 - tar xvf filename.tar
- tar.gz 한번에 풀기
 - tar xvzf filename.tar.gz

SSH (Secure Shell)

- ssh

- 원격 컴퓨터에 안전하게 접속하여 필요한 작업을 할 수 있게 해주는 보안 로그인 프로토콜/인터페이스
- 리눅스에서는 주로 OpenSSH 클라이언트가 사용(실행)된다
- ssh penguin.example.net
 - 암호를 입력하면 penguin.example.net이라는 원격 컴퓨터에 로그인하여 셸 프롬프트가 나타난다

- scp

- 컴퓨터 사이에 암호화된 명령을 통하여 파일을 전송하는 데 사용
- scp Wheelseminar.ppt username@penguin.example.net:/uploads/
 - Wheelseminar.ppt 파일을 원격 컴퓨터 penguin.example.net 상의 uploads 디렉토리로 전송

와일드카드 문자 (Wildcard Character)

- 하나 이상의 다른 문자를 상징하는 특수문자
 - * (별표, asterisk): 문자열에 있는 0개 이상의 문자를 상징
 - ? (물음표, question mark): 어떤 한 개의 문자를 상징
- run* – run, runs, running, runner, runners...
- ?at – bat, cat, eat, fat, hat, rat...
- 리눅스에서는 파일 경로명 등에 요긴하게 쓸 수 있다
 - `rm *.log` – 모든 로그 파일을 삭제한다
 - `grep -r -i -n myfunction *` – 현재디렉토리와 하위디렉토리에서 대소문자 구별 없이 myfunction이라는 내용이 들어있는 파일을 찾아서 그 행과 번호를 출력

파이프 (Pipe) |

- 두 개 이상의 명령어를 '파이프로 연결'
- 한 명령어의 output이 다른 명령어의 input으로 연결된다.
- `ls -l | less`
 - `ls -l` 명령어의 출력이 `less` 명령어의 입력으로 연결된다
 - 현재 작업 디렉토리에 있는 파일 상세정보를 한 페이지씩 넘겨서 볼 수 있다
- `ps | grep chrome`
 - 실행중인 프로세스 목록 중에 `chrome`이 들어가는 행을 출력
 - `chrome`의 프로세스 ID를 한눈에 찾을 수 있다
- `yes`
 - 'y'를 끝없이 출력하는 명령어
 - `yes | rm *.txt`
 - 파일을 정말로 삭제하겠냐는 확인 메시지에 끊임없이 y가 입력되므로 해당 메시지를 무시하게 된다 (= `rm -f`)

리디렉션 (Redirection)

- 셸에서는 표준입력(stdin), 표준출력(stdout), 표준에러출력(stderr)이 이미 지정해진 파일이나 장치로 지정되어 있다.
 - stdin – 키보드 입력
 - stdout, stderr – 모니터에 출력되는 셸 터미널
- 이를 특별한 연산자 <, >, 2>를 이용하여 재지정(redirect)할 수 있다.
 - mail mother < my_typed_letter
 - cat file1 file2 > file3
 - echo "this is a new line" >> existingfile.txt
 - myprogram 2> errorsfile

명령 히스토리 (History Expansion)

- 명령어를 입력하면 바로 없어지는 게 아니라 히스토리로 저장된다
 - Ctrl+R을 누르고 찾고자 하는 단어를 입력하면 내가 입력했던 명령어 중에 그 단어가 들어간 명령어를 탐색할 수 있다.
 - 엔터로 바로 입력하거나, 왼쪽/오른쪽 방향키로 내용을 수정하면 된다.
- 특정 단어로 시작하는 히스토리
 - [!특정단어]를 입력하여 지난 번에 입력한 특정 단어로 시작하는 명령어를 실행할 수 있다.

Reference

- [http://www.utm.edu/organizations/tsa/Books/\(OReilly\)%20Running%20Linux,%204th%20Edition.pdf](http://www.utm.edu/organizations/tsa/Books/(OReilly)%20Running%20Linux,%204th%20Edition.pdf)
- <http://www.gnu.org/software/bash/manual/bashref.html>
- <http://www.wikipedia.org>
- <http://linux.about.com/>
- <http://www.tldp.org/LDP/intro-linux/html/>
- <http://stone.backrush.com/sunfaq/h00023.html>
- http://www.fis.unipr.it/pub/linux/redhat/9/en/doc/RH-DOCS/rhl-cg-ko-9/s1_openssh-client-config.html
- http://radiocom.kunsan.ac.kr/lecture/unix_cmd/redirectation.html
- <http://www.codecoffee.com/tipsforlinux/articles2/042.html>
- <http://parkmino45.blog.me/140130909136>
- 앞선 세미나 자료, 선배님들의 조언