

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots. The lines are thin and gray, creating a subtle background pattern.

# 리눅스 기초


A decorative network diagram in the bottom-right corner, similar to the one in the top-left, featuring a complex web of interconnected nodes and lines. Some nodes are highlighted with blue circles, and others with blue dots. The lines are thin and gray, creating a subtle background pattern.

## 이 슬라이드에서...

- ◎ 가상화
- ◎ 리눅스 소개
- ◎ 필수 개념과 명령어 소개
- ◎ Shell스크립트 프로그래밍



# 가상머신



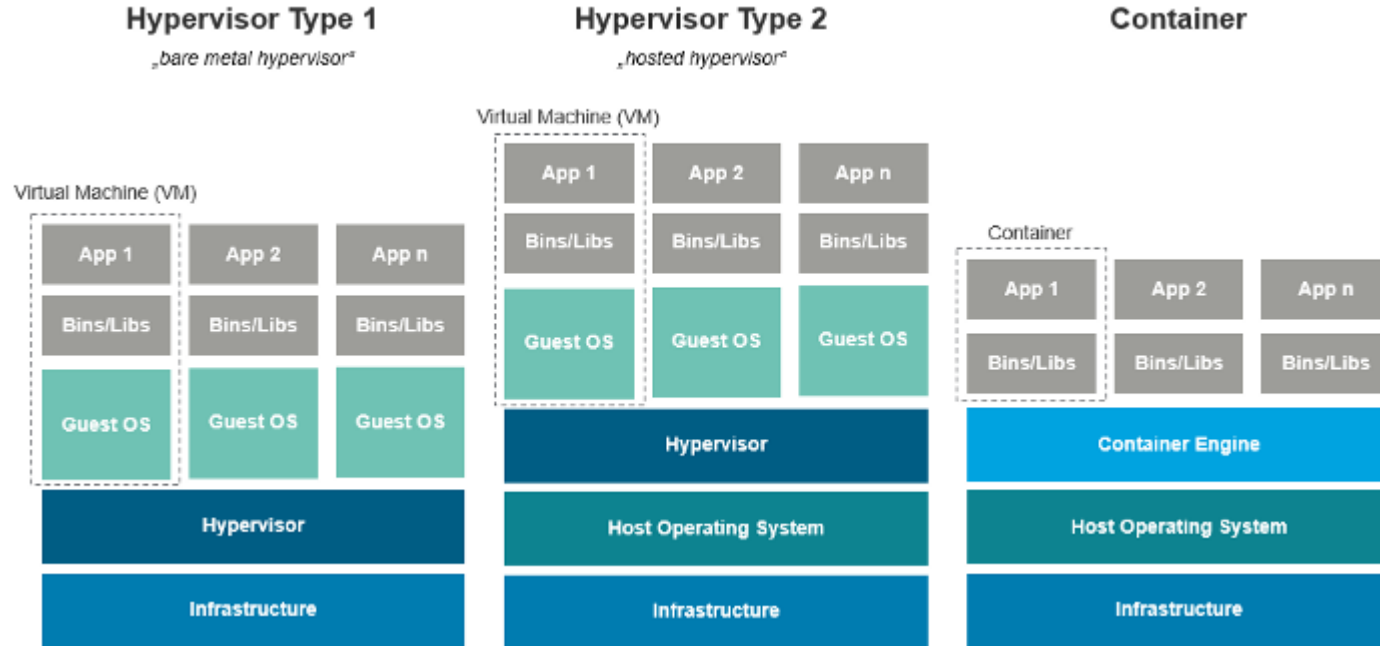
# Operating System(운영체제)



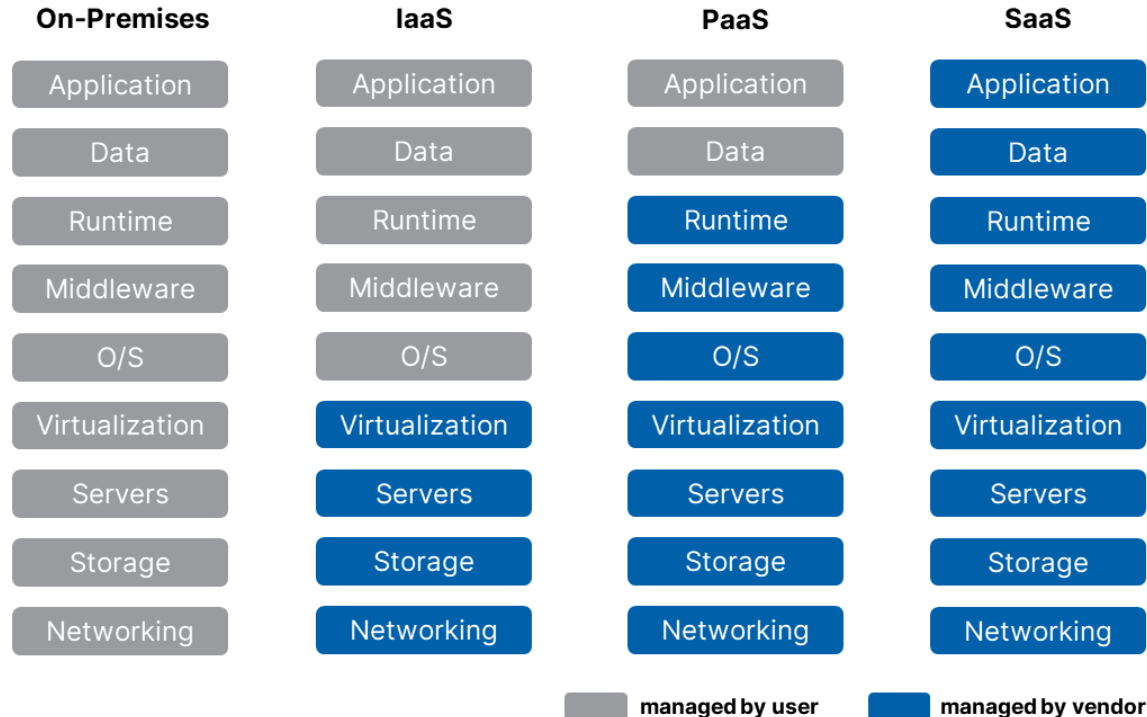
**Linux**



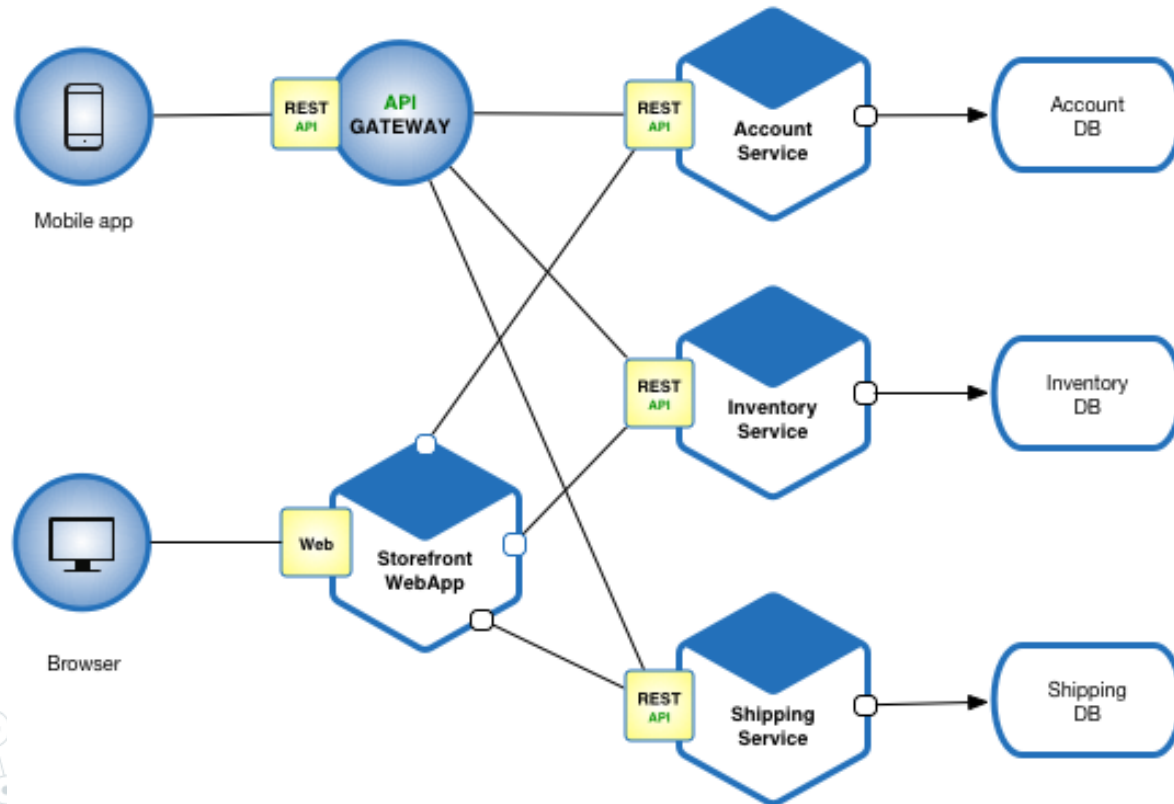
# VM vs Container



# Types of cloud computing services



# Microservices



# CPU: X86 vs ARM

- ◎ Arm is RISC (Reduced Instruction Set Computing) based
  - "알아듣기 쉬운 말로 일 설명"
  - Exynos, M1, PowerPC, etc
- ◎ Intel (x86) is CISC (Complex Instruction Set Computing) based
  - "적은 말로 많은 일 설명 "
  - Intel, AMD, mos6502(Apple ][), etc
  - [https://en.wikipedia.org/wiki/X86\\_instruction\\_listings](https://en.wikipedia.org/wiki/X86_instruction_listings)

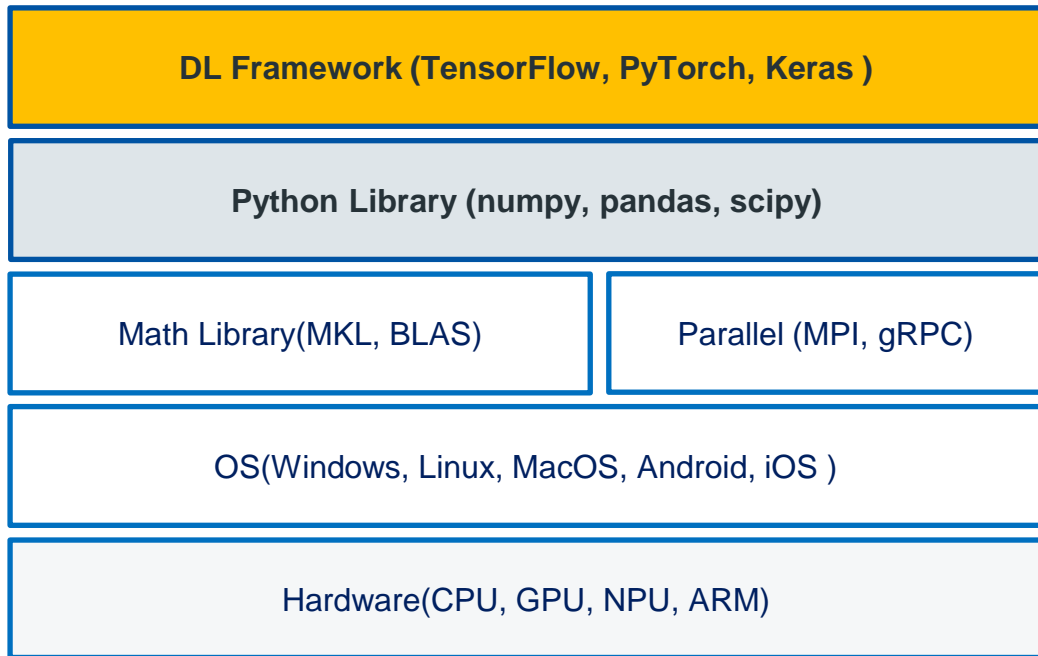
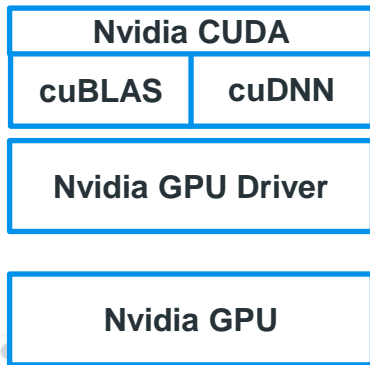


# CISC vs RISC

- ◎ CISC 접근 방식은 프로그램당 명령어 수를 최소화하려고 시도하여 명령어당 사이클 수를 희생합니다.
- ◎ RISC는 반대로 프로그램당 명령어 수를 희생(증가)시키면서 명령어당 사이클을 줄입니다.

CISC	RISC
하드웨어 강조	소프트웨어 강조
다중 사이클, 복합 명령어	단일 사이클, 축소 명령어만
메모리 to 메모리: "LOAD" 및 "STORE" 명령어에 통합된	레지스터 to 레지스터: "LOAD" 및 "STORE" 는 독립적 명령어
작은 코드 크기, 높은 초당 사이클	사이클 낮은 초당 사이클, 큰 코드 크기
복잡한 명령어를 저장하는 데 트랜지스터 사용(비교적)	메모리 레지스터의 더 많은 트랜지스터를 사용

# 딥러닝 프레임워크



A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines are thin and grey, creating a mesh-like structure that extends from the top-left towards the center of the slide.

1.

# 리눅스 소개

## 리눅스란

- ◎ 대부분 윈도우나 맥을 사용해 본 적이 있을 것인데 리눅스도 이들과 비슷한 운영 체제
- ◎ 컴퓨터라는 하드웨어에서 다양한 애플리케이션을 돌리기 위한 기본 소프트웨어
- ◎ 리눅스는 일반적인 데스크톱뿐 아니라 임베디드, 서버 등 다양한 분야에서 널리 사용되고 있음

# 리눅스의 장점

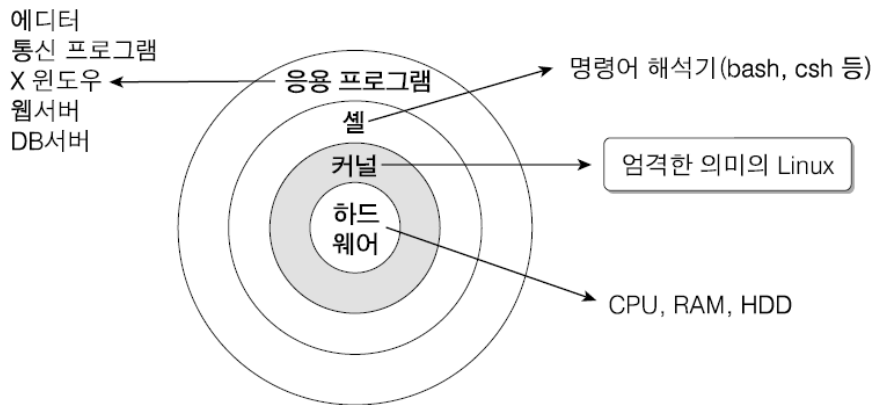
- ◎ 높은 품질의 다양한 소프트웨어를 리눅스에서 돌릴 수 있음(아파치 http 서버나 MySQL 데이터베이스 등)
- ◎ 스크립트를 통해 많은 부분을 자동화할 수 있어 운영에 편리함
- ◎ 브라우저에서 요청을 받아들이는 웹 서버, 비즈니스 로직을 수행하는 애플리케이션 서버, 데이터를 저장하는 데이터베이스의 운영 체제로 리눅스를 사용하는 것이 일반적이기 때문임
- ◎ 개발 환경 구축이 용이하며 비용이 발생하지 않음
- ◎ 오픈 소스여서 운영 체제의 내부 동작을 확인할 수 있음

## 리눅스의 단점

- ◎ 리눅스는 윈도우나 맥보다 상용 애플리케이션이 많지 않음
- ◎ 마이크로소프트의 워드나 엑셀 파일을 편집하려면  
리브레오피스(LibreOffice) 같은 프로그램을 사용해야 함
- ◎ 이미지, 음악, 영상 등 멀티미디어를 편집하는 소프트웨어도 부족함
- ◎ 한글 대응이 부족

## 리눅스의 개요

- ◎ 리눅스 = 무료 유닉스(1969 AT&T Unix)
- ◎ 1991년 '리누스 토르발스'가 버전 0.01을 최초로 작성
- ◎ 1992년 0.02 버전을 공개하면서 시작됨
- ◎ 리누스 토르발스는 커널(Kernel)만 개발함
- ◎ 배포판의 구성



## GNU 프로젝트

- ◎ 1984년에 리처드 스톨만(Richard Stallman)에 의해서GNU 프로젝트가 시작
- ◎ 목표는 '모두가 공유할 수 있는 소프트웨어'를 만드는 것
- ◎ 리처드 스톨만은 1985년에 자유 소프트웨어 재단(FSF, Free Software Foundation)을 설립
- ◎ 목표는 GNU 프로젝트에서 제작한 소프트웨어를 지원함으로써 컴퓨터 프로그램의 복제, 변경, 소스 코드의 사용에 대한 제한을 철폐하는 것
- ◎ GPL(General Public License)을 따름. 이 라이선스는 자유 소프트웨어(Free Software)의 수정과 공유의 자유를 보장함
- ◎ 프리웨어(Freeware, 무료 소프트웨어)라는 개념을 뛰어넘어서 진정한 자유(Freedom)에 대한 개념
- ◎ 자유 소프트웨어는 심지어 무료로 얻은 소프트웨어를 유상으로 판매할 자유도 보장



## 커널

◎ <http://www.kernel.org> 에서 최신버전을 무료로 다운로드

◎ 커널 변천사

커널 버전	0.01	1.0	2.0	2.2	2.4	2.6	3.0	3.2	3.4	3.10
발표 연도	1991	1994	1996	1999	2001	2003	2011	2012	2012	2013

◎ 커널 버전의 의미 (예: 3.17.4 )

- 3는 주 버전 (Major Version)
- 17은 부 버전 (Minor Version)
- 4는 패치 버전 (Patch Version)

◎ 배포판에 포함된 기본 커널을 사용자가 직접 최신의 커널로 업그레이드할 수 있음 (커널 업그레이드)

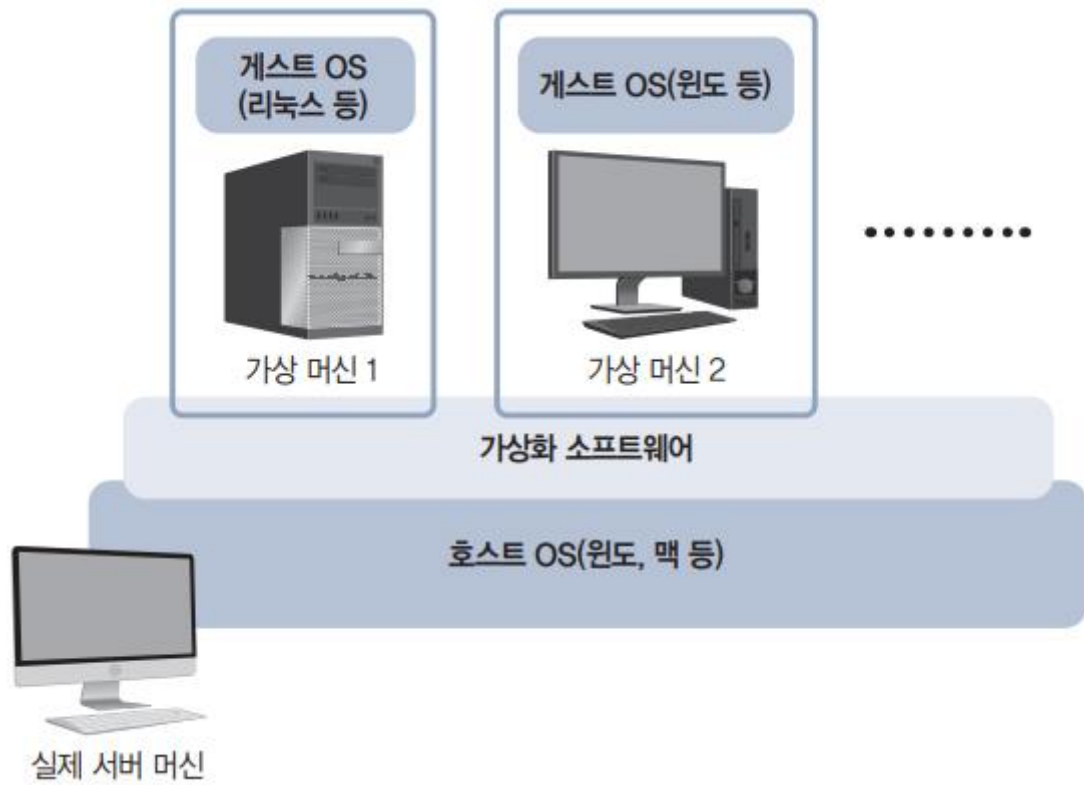
## 라즈비안 파이 OS(Raspbian Pi OS)

- ◎ 라즈베리 파이 재단이 개발한 라즈베리 파이 전용 운영 체제
- ◎ 과거에는 라즈비안(영어: Raspbian)이라는 이름을 사용
- ◎ 라즈베리 파이 OS는 마이크 톰슨(Mike Thompson)과 피터 그린(Peter Green)이 독립 프로젝트로 개발
- ◎ 최초 빌드는 2012년 6월에 완성
- ◎ 라즈베리 파이 OS는 라즈베리 파이 계열의 저성능 ARM CPU에 최적화
- ◎ 라즈베리 파이 OS는 픽셀(PIXEL, Pi Improved Xwindows Environment, Lightweight)을 자체 주 데스크톱 환경으로 사용
- ◎ 최신 버전 기준으로 가벼운 버전의 크로미엄을 포함

## 가상화 소프트웨어 위의 리눅스 환경

- ◎ 리눅스를 익히려면 실습을 하면서 그 동작을 확인해 보아야 함
- ◎ 리눅스를 설치하는 첫 번째 방법은 리눅스를 컴퓨터에 직접 설치하는 것
- ◎ 윈도우가 설치된 컴퓨터의 하드 디스크를 분할하여 리눅스를 추가로 설치하는 것
- ◎ 이는 간단한 작업이 아니며 자칫 잘못하면 현재 환경이 망가질 수도 있음
- ◎ 가상화 소프트웨어를 사용해 컴퓨터에 가상 컴퓨터, 즉 가상 머신에 리눅스 환경을 구축해 보자

# 가상화 소프트웨어의 개념



## 라즈비안 파이 OS 실습

- ◎ 가상화 소프트웨어 오라클 VM 버추얼박스 사용
  - "**Raspian on Virtualbox.pdf**" 를 참고해주세요
  - "01\_라즈베리파이 첫시작.pdf"
- ◎ 라즈베리 파이용 SD카드에 라즈비안 파이 OS를 설치합니다
  - "라즈베리 파이 Linux 준비.pdf"를 참고해주세요
  - "01\_라즈베리파이 첫시작.pdf"

A decorative network diagram in the top-left corner, consisting of a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels or types of connectivity. The lines are thin and gray, creating a mesh-like structure.

# 2.

## 로그인, 쉼다운, sudo

종료와 재부팅은 sudo필요  
\$sudo su -

## 시작과 종료

### ◎ 종료하는 방법

- ① 바탕 화면의 [사용자 이름] → [컴퓨터 끄기] → <컴퓨터 끄기>
- ② 터미널/콘솔에서 시스템 종료 명령 입력  
"shutdown -P now" , "halt -p" , "init 0"

### ◎ 시스템 재부팅

- ① 바탕 화면의 [사용자 이름] → [컴퓨터 끄기] → <다시 시작>
- ② 터미널/콘솔에서 시스템 재부팅 명령 입력  
"shutdown -r now" , "reboot" , "init 6"

### ◎ 로그아웃

- ① 바탕 화면의 [사용자 이름] → [로그아웃]
- ② 터미널/콘솔에서 시스템 종료 명령 입력  
"logout" 또는 "exit"

## shutdown [option...] [time] [wall...]

옵션	설명
-H	시스템을 종료
-P	시스템을 종료하고 전원을 끄
-r	시스템 재시작
-h	시스템을 종료하고 전원을 끄(-P와 같다)
-c	예약된 셧다운 명령을 취소
-k	실제로 셧다운 명령을 수행하지 않고 경고 메시지만 전달
TIME	몇 분 후나 종료할 시간을 지정
WALL	시스템 종료 시 시스템에 접속하여 사용하는 사용자들에게 보낼 메시지



## 런 레벨(Runlevel)

- ◎ 'init' 명령어 뒤에 붙는 숫자를 런레벨RunLevel이라고 부른다.

런레벨	영문 모드	설명	비고
0	Power Off	종료 모드	
1	Rescue	시스템 복구 모드	단일 사용자 모드
2	Multi-User		사용하지 않음
3	Multi-User	텍스트 모드의 다중 사용자 모드	
4	Multi-User		사용하지 않음
5	Graphical	그래픽 모드의 다중 사용자 모드	
6	Reboot		

- ◎ 런레벨 모드를 확인하려면 `/lib/systemd/system` 디렉터리의 `runlevel#.target` 파일을 확인

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with solid centers and others with dashed outlines. The lines are thin and gray, creating a mesh-like structure.

# 3. 셀

## 3.1 셸과 명령어

- 셸과 명령어
  - 리눅스에 로그인했다면 곧바로 명령어를 입력해 보자
  - 다음과 같이 \$ 기호 뒤에 date 명령어를 입력하고 **Enter**를 눌러 보자

• date 명령어 실행

```
ldk@ldk-VirtualBox:~$ date  
2021. 02. 13. (토) 21:52:59 KST
```

- 현재 날짜와 시간이 출력
- date 명령어는 현재 시간을 출력하거나 설정하는 명령어

## 3.1 셸과 명령어

- 셸과 명령어
  - 다른 명령어도 사용해 보자
  - echo 명령어는 인자로 지정한 문자열을 출력
  - 여기서는 인자로 Hello를 지정해 보자

● echo 명령어 실행

```
ldk@ldk-VirtualBox:~$ echo Hello  
Hello
```

## 3.1 셸과 명령어

- 셸과 명령어
  - 이외에도 명령어가 다양함
  - 처음 설치할 때부터 수십 개 이상의 명령어가 포함되어 있음
  - 이 모든 명령어를 외울 필요는 없음
  - 필수적인 명령어들을 익힌 뒤에 필요할 때마다 하나씩 익히면 됨
  - 필요한 작업에 딱 맞는 명령어가 없다면 기존 명령어들을 조합하여 사용하면 됨
  - 리눅스는 단순한 명령어들을 조합하여 복잡한 작업을 처리할 수 있도록 설계

## 3.1 셸과 명령어

- 에러에 대해서
  - 존재하지 않는 명령어를 입력하면 어떻게 될까?
  - 예를 들어 `abcxyz`라는 명령어를 입력해 보자

- 에러 메시지의 예

```
ldk@ldk-VirtualBox:~$ abcxyz  
abcxyz: 명령을 찾을 수 없습니다
```

- 에러 메시지가 출력
- `abcxyz`라는 명령어가 없어서 찾을 수 없다는 메시지

## 3.1 셸과 명령어

- 에러에 대해서
  - 영문 리눅스에서는 다음과 같이 표시
    - 영문 환경에서의 에러 메시지

```
ldk@ldk-VirtualBox:~$ abcxyz  
abcxyz: command not found
```

## 3.1 셸과 명령어

- 에러에 대해서
  - 리눅스 환경에서 실습하다 보면 오타 등으로 인해 다양한 에러를 만나게 될 것
  - 에러를 만나면 당황하지 말고 침착하게 표시된 에러 메시지를 읽으면서 원인을 파악해 보기 바람
  - 에러 메시지에는 에러가 발생한 원인에 대한 실마리가 담겨 있음
  - 에러 메시지를 잘 읽고 그 내용에 맞게 잘 대처하는 것이 리눅스를 익히는 데 매우 중요한 자세



## 3.1 셸과 명령어

- 셸의 역할과 리눅스 커널
  - 이 장의 목표는 셸이 무엇인지를 알아보는 것
  - 이를 위해 리눅스 내부에서 명령어가 실행되는 과정을 살펴보자
  - 앞서 date나 echo 같은 명령어를 실행해 보았음
  - 이때 리눅스의 내부에서는 다음과 같은 일이 일어남

1 | 키보드로 입력한 date 문자열을 받아들임

2 | date 명령어를 찾음

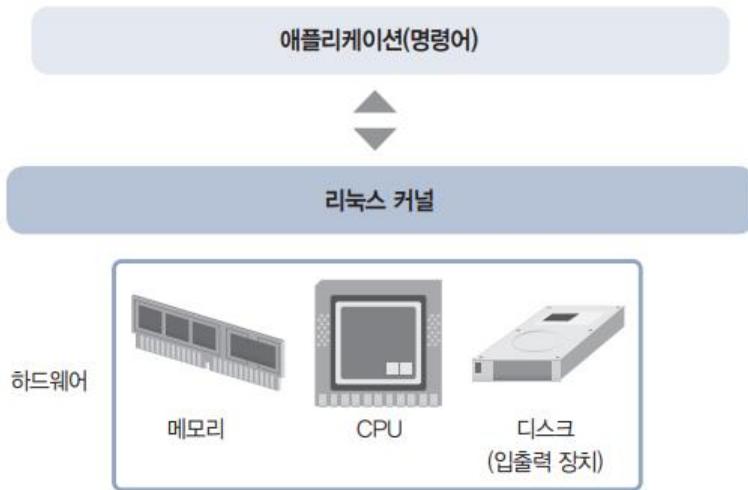
3 | 발견한 명령어를 실행

4 | 실행한 결과로 얻은 문자열을 화면에 표시

## 3.1 셸과 명령어

- 셸의 역할과 리눅스 커널
  - 이 중에서 3번은 리눅스의 본체인 커널이 수행
  - 커널은 운영 체제의 중심에서 CPU나 메모리 같은 하드웨어를 관리하면서 명령어를 실행하고 프로세스를 관리

### ▼ 그림 3-1 리눅스 커널과 하드웨어



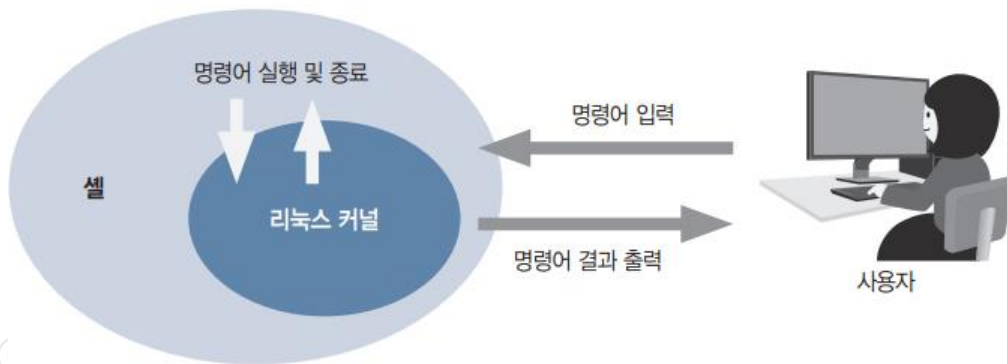
## 3.1 셸과 명령어

- 셸의 역할과 리눅스 커널
  - 앞서 사용자는 date라는 문자열을 키보드로 입력하고 `Enter`를 눌렀음
  - 엄밀히 말하면 사용자가 직접 리눅스 커널을 조작한 것은 아님
  - 리눅스에서는 사용자가 커널을 직접 조작할 수 없게 되어 있기 때문에 둘 사이에서 명령어를 받아들이고 커널의 실행 결과를 출력하는 소프트웨어가 필요함
  - 이 역할을 수행하는 소프트웨어가 바로 셸
  - 즉, 셸은 커널의 인터페이스에 해당

## 3.1 셸과 명령어

- 셸의 역할과 리눅스 커널
  - 앞 예에서는 사용자가 셸에 date 문자열을 입력
  - 셸은 date 명령어를 찾아서 리눅스 커널에게 실행을 의뢰함
  - 리눅스 커널이 명령을 실행하면 셸은 그 결과를 전달받아 사용자의 화면에 출력

▼ 그림 3-2 사용자와 리눅스 커널 사이의 소통 창구인 셸



## 3.2 프롬프트

- 프롬프트
  - 앞서 명령어를 입력할 때 다음과 같은 문자열을 보았을 것

### ▼ 그림 3-3 프롬프트

#### ● 우분투 프롬프트

```
user@hostname:~$
```

- 이를 셸의 프롬프트(prompt)라 함
- 프롬프트는 사용자에게 어떤 결정을 내리도록 한다는 의미
- 즉, 셸이 사용자에게 명령어를 받아들일 준비가 되었음을 나타낸다고 보면 됨
- 셸의 프롬프트는 커스터마이징할 수 있는데 우분투에서는 기본적으로 위와 같이 표시

## 3.2 프롬프트

- 프롬프트 기호
  - 지금부터 프롬프트를 표시할 때는 다음과 같이 일반 사용자는 \$, 슈퍼 사용자(root)는 # 만을 표시

### ● 일반 사용자의 프롬프트

\$ <명령어>

### ● 슈퍼 사용자의 프롬프트

# <명령어>

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are solid grey and others are hollow with a grey outline. The lines connecting them are thin and grey, creating a dense, organic structure.

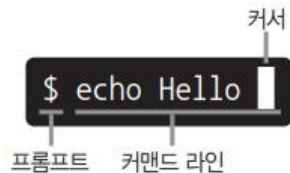
4.

# 필수 개념과 명령어

# 커맨드 라인 편집

- 커맨드 라인 편집
  - 셸에서 프롬프트 기호(\$) 뒤에 명령어를 입력하는 부분을 커맨드 라인(행)이라고 함

## ▼ 그림 4-1 프롬프트와 커맨드 라인



## ▼ 표 4-2 커서 이동 단축키

단축키	내용
Ctrl + b	커서를 한 문자 뒤로 이동합니다.
Ctrl + f	커서를 한 문자 앞으로 이동합니다.
Ctrl + a	커서를 맨 앞으로 이동합니다.
Ctrl + e	커서를 맨 뒤로 이동합니다.



## 자동 완성과 (명령 이력)히스토리

- ◎ 자동 완성이란 파일명의 일부만 입력한 후에 Tab키를 눌러 나머지 파일명을 자동으로 완성하는 기능을 말함.

예) \$ ec  
\$ echo

를 입력하고 [Tab키]를 입력  
자동완성

자동 완성기능은 빠른 입력효과도 있지만, 파일명이나 디렉터리가 틀리지 않고 정확하게 입력되는 효과도 있으므로 자주 활용된다.

- ◎ 도스 키란 이전에 입력한 명령어를 상/하 화살표 키를 이용해서 다시 나타내는 기능을 말함.

## 〈실습〉 자동 완성과 도스 키

- 실습목표
  - 자동 완성 기능과 도스 키 기능을 익힌다.
  - history 명령어의 기능을 확인한다.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ history  
1 make -j $(nproc)  
2 cd ~  
3 cd ..  
4 ls  
5 zip -r opencv-4.4.0.zip opencv-4.4.0/  
6 tar -cvf opencv-4.4.0.tar opencv-4.4.0/  
7 tar -cvf opencv_contrib-4.4.0.tar opencv_contrib-4.4.0/  
8 sudo make install && sudo ldconfig  
9 cd opencv-4.4.0/  
10 cd build  
11 sudo make install && sudo ldconfig  
12 sudo apt-get update  
13 sudo apt-get upgrade  
14 pip3 install opencv-contrib-python
```

### 명령 이력 관련 단축키

명령어	내용
history	명령 이력
!!	이전 명령
!9	9번 전 명령

# 명령 이력

## 명령 이력 관련 단축키

단축키	내용
<b>Ctrl</b> + <b>p</b> 혹은 <b>↑</b>	바로 전 명령으로 이동합니다.
<b>Ctrl</b> + <b>n</b> 혹은 <b>↓</b>	다음 명령으로 이동합니다.
<b>Ctrl</b> + <b>r</b>	이력을 검색합니다.

## 증분 검색

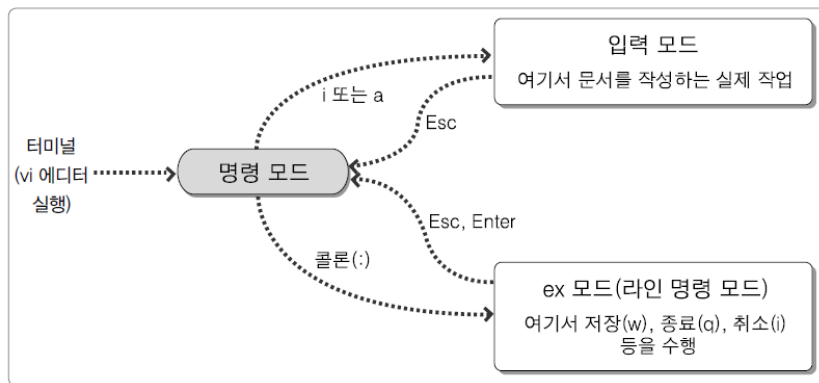
단축키	내용
(문자 입력)	문자를 하나씩 입력할 때마다 검색을 수행합니다.
<b>Ctrl</b> + <b>r</b>	한 개 이전의 검색 결과로 이동합니다.
<b>Enter</b>	현재 검색 결과를 실행합니다.
<b>Esc</b>	현재 검색 결과를 실행하지 않은 채 커맨드 라인으로 복귀합니다.
<b>Ctrl</b> + <b>g</b>	검색 결과를 지우고 프롬프트로 복귀합니다.

vi는 자주 사용해야 할 기능이므로  
반드시(?) 익혀야 한다.

## 에디터 사용

### <실습> 에디터를 사용하자

- 실습목표
  - gedit의 기본적인 사용법을 익힌다.
  - vi의 사용법을 연습한다. `$ vi file.name`
- vi 에디터 사용법 개요도



## vi 기능 요약

### ◎ 명령모드 → 입력모드

i	현재 커서의 위치부터 입력([I])	I	현재 커서 줄의 맨 앞에서부터 입력([Shift] + [I])
a	현재 커서의 위치 다음 칸부터 입력([A])	A	현재 커서 줄의 맨 마지막부터 입력([Shift] + [A])

### ◎ 명령 모드에서 커서를 이동

h	커서를 왼쪽으로 한 칸 이동([←]와 같은 의미, [H])	j	커서를 아래로 한 칸 이동([↓]와 같은 의미, [J])
k	커서를 위로 한 칸 이동([↑]와 같은 의미, [K])	l	커서를 오른쪽으로 한 칸 이동([→]와 같은 의미, [L])

### ◎ 명령 모드에서 삭제, 복사, 붙여넣기

x	현재 커서가 위치한 글자 삭제([Del]과 같은 의미, [X])	X	현재 커서가 위치한 앞 글자 삭제([BackSpace]와 같은 의미, [Shift] + [X])
dd	현재 커서의 행 삭제([D] 연속 두 번 입력)	숫자 dd	현재 커서부터 숫자만큼의 행 삭제(숫자 다음 [D] 연속 두 번 입력)
yy	현재 커서가 있는 행을 복사([Y] 연속 두 번 입력)	숫자 yy	현재 커서부터 숫자만큼의 행을 복사(숫자 다음 [Y] 연속 두 번 입력)
p	복사한 내용을 현재 행 이후에 붙여 넣기([P])	P	복사한 내용을 현재 행 이전에 붙여 넣기([Shift] + [P])

1. 저장만 : w
2. 종료만 : q
3. 저장 후 종료 : wq

## Vi 파일 열기와 저장하기

- ◎ 파일 열기
  - vi newfile1.txt
- ◎ 저장하기
  - :w
- ◎ Vi에서 나오기
  - :q

명령어	내용
:q	Vim 종료
:w	저장
:w <파일 이름>	파일 이름을 지정하여 저장
:q!	저장하지 않고 Vim 종료

# Vi 파일 편집

## \* 텍스트파일 복사

- cp /etc/crontab .

## ◎ Vi 파일 편집

- vi crontab

## ◎ 커서 이동

h	커서를 왼쪽으로 한 칸 이동(←)와 같은 의미, [H])	j	커서를 아래로 한 칸 이동(↓)와 같은 의미, [J])
k	커서를 위로 한 칸 이동(↑)와 같은 의미, [K])	l	커서를 오른쪽으로 한 칸 이동(→)와 같은 의미, [L])

## ◎ 문자 삭제

- x

```
# For details see man 4 crontabs
```



[x]를 눌러 문자 삭제

```
# or details see man 4 crontabs
```

## Vi 파일 편집

### ◎ 문자 입력

- i

```
# █ details see man 4 crontabs
```



i를 누르고 j o b을 입력

```
# For█ details see man 4 crontabs
```

- `[Esc]` 로 입력 모드 종료



## Vi 파일 편집

### ◎ 문자 입력

- **i**를 통해 입력 모드로 전환한 경우에는 입력하는 글자가 커서의 왼쪽에 추가
- 이와 반대로 커서의 오른쪽에 문자를 입력하고 싶은 경우에는 **a**를 사용
- 다음은 **a**를 눌러 입력 모드로 전환한 뒤 문장의 끝에 !를 추가한 예
- 입력 모드를 종료하는 방법은 마찬가지로 **Esc**를 누르면 됨

```
# For details see man 4 crontabs
```

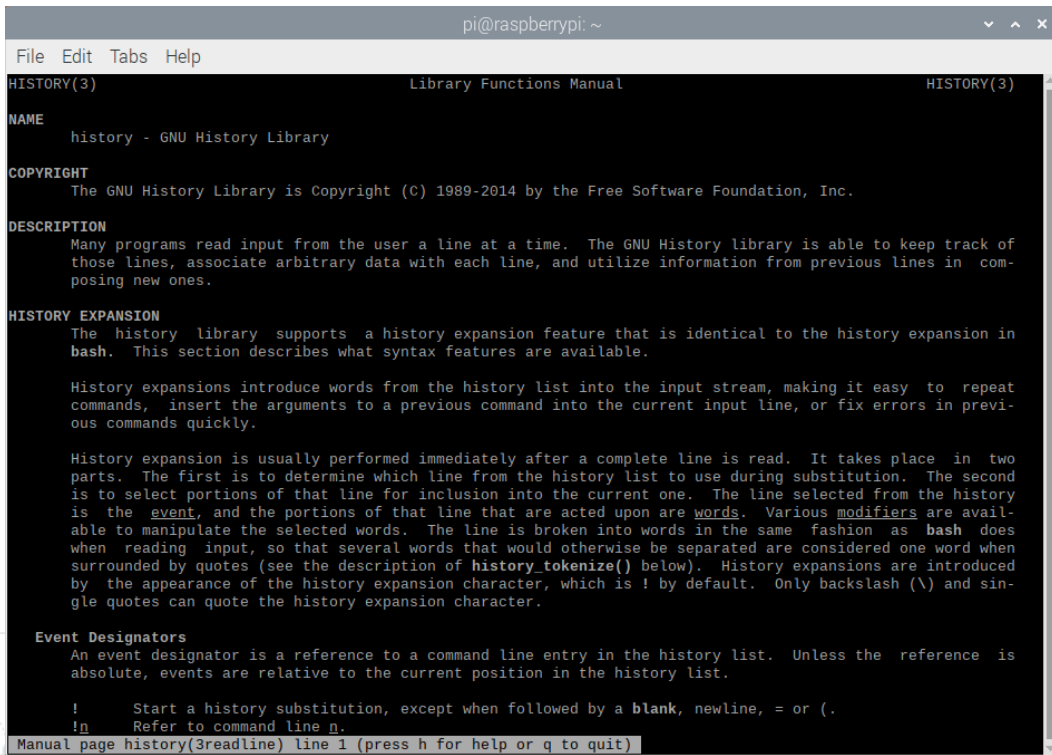


**a**를 누르고 **i**를 입력

```
# For details see man 4 crontabs!
```

## 도움말 사용법

### ◎ “man 명령어”를 사용하면 도움말 출력



```
pi@raspberrypi: ~
File Edit Tabs Help
HISTORY(3) Library Functions Manual HISTORY(3)

NAME
  history - GNU History Library

COPYRIGHT
  The GNU History Library is Copyright (C) 1989-2014 by the Free Software Foundation, Inc.

DESCRIPTION
  Many programs read input from the user a line at a time.  The GNU History library is able to keep track of those lines, associate arbitrary data with each line, and utilize information from previous lines in composing new ones.

HISTORY EXPANSION
  The history library supports a history expansion feature that is identical to the history expansion in bash.  This section describes what syntax features are available.

  History expansions introduce words from the history list into the input stream, making it easy to repeat commands, insert the arguments to a previous command into the current input line, or fix errors in previous commands quickly.

  History expansion is usually performed immediately after a complete line is read.  It takes place in two parts.  The first is to determine which line from the history list to use during substitution.  The second is to select portions of that line for inclusion into the current one.  The line selected from the history is the event, and the portions of that line that are acted upon are words.  Various modifiers are available to manipulate the selected words.  The line is broken into words in the same fashion as bash does when reading input, so that several words that would otherwise be separated are considered one word when surrounded by quotes (see the description of history_tokenize() below).  History expansions are introduced by the appearance of the history expansion character, which is ! by default.  Only backslash (\) and single quotes can quote the history expansion character.

Event Designators
  An event designator is a reference to a command line entry in the history list.  Unless the reference is absolute, events are relative to the current position in the history list.

  !      Start a history substitution, except when followed by a blank, newline, = or (.
  !n     Refer to command line n.

Manual page history(3readline) line 1 (press h for help or q to quit)
```

# Section

section 번호	설명
1	실행 가능한 프로그램 또는 셸 명령
2	시스템 호출(커널이 제공하는 기능)
3	C 라이브러리 호출 (프로그램 라이브러리 내의 기능)
4	디바이스와 관련된 특수 파일 (보통 /dev 디렉터리 안에 있음)
5	파일 형식 및 규칙(예) /etc/passwd)
6	게임
7	기타 (매크로 패키지 및 규칙 포함) (예) man (7), groff (7)
8	시스템 관리 명령 (일반적으로 root 관리자만 해당)
9	커널 루틴 [비표준]

man -f [command]  
보유한 섹션을 확인

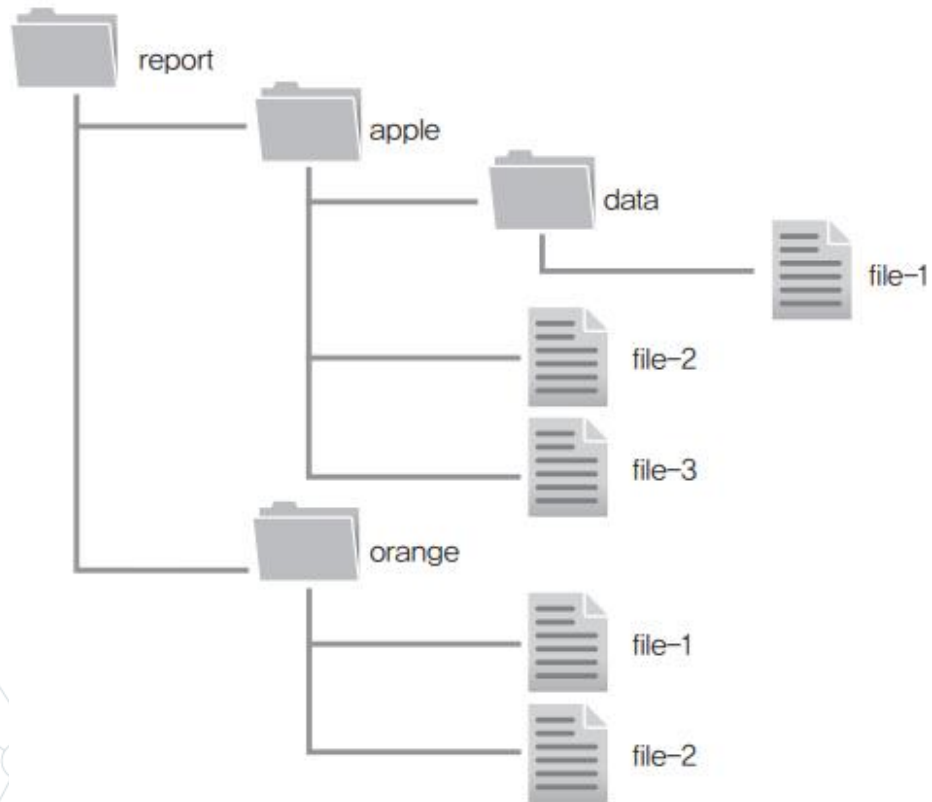
## date

명령어	설명
date "+%B %d"	오늘 날짜를 로케일의 완전한 월 이름과 일 (01..31)로 표시
date "+%D %r"	오늘 날짜 (mm/dd/yy)와 시간, 12-시간제 (hh:mm:ss [AM PM])
date --date "2 days ago"	2일 전의 날짜 출력
date --date "3 months 3days"	3개월 3일 후 날짜 출력
date --date "next year 25 Dec"	다음연도 크리스마스 날짜 출력

## cal

달력을 출력

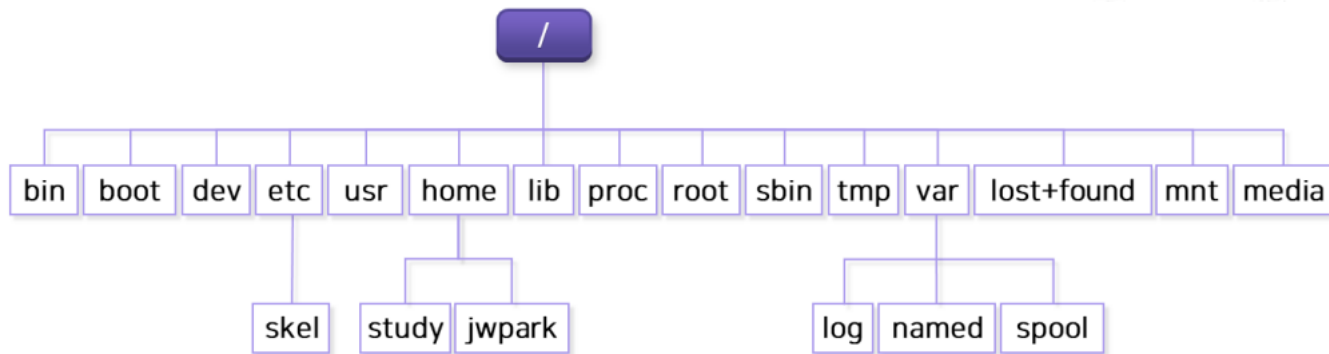
## 디렉터리와 파일의 예



# File System

Filesystem Hierarchy Standard

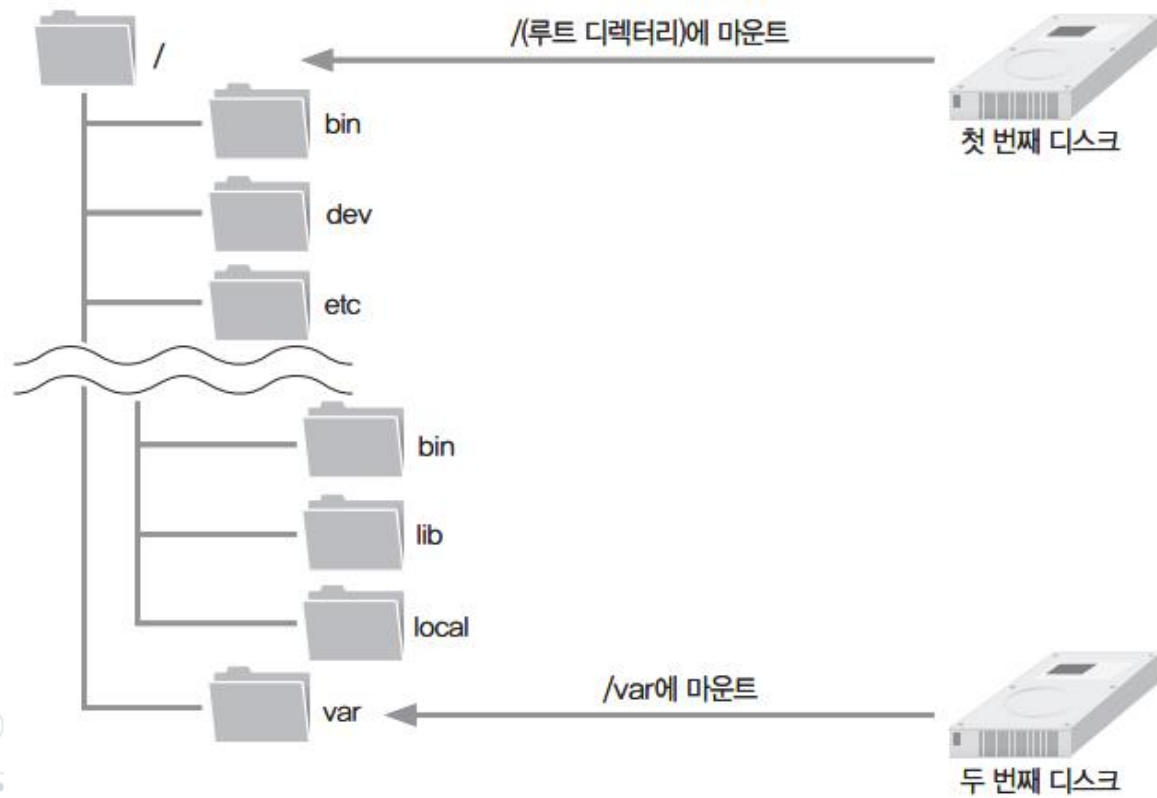
URL <http://www.pathname.com/fhs/>



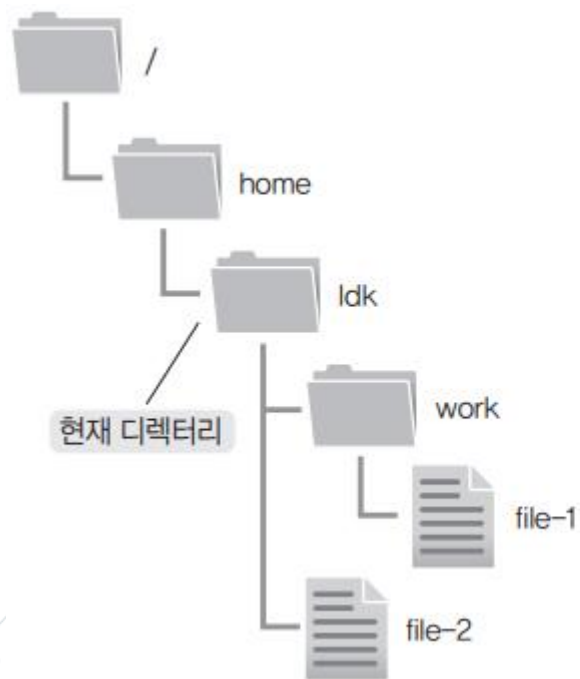
사용자들이 파일에 정보를 저장하고 검색할 수 있도록 되어있음

- ◎ 일반파일
  - 디렉터리안에 같은 이름의 파일이 존재할 수 없음
  - Shell과 관계되는 특수문자들을 사용할 수 없음
- ◎ 디렉터리파일
  - 다른 파일들과 디렉터리들을 저장하는 논리적 영역
- ◎ 특수파일
  - 입/출력장치를 접근하고 관리하는 채널에 대한 정보파일

# 디스크가 여러 개 있어도 디렉터리 트리는 하나



## 절대 경로와 상대 경로



상대 경로	절대 경로
../..	/
..	/home
.	/home/ldk
work 혹은 ./work	/home/ldk/work
work/file-1 혹은 ./work/file-1	/home/ldk/work/file-1
file-2 혹은 ./file-2	/home/ldk/file-2



## 리눅스 기본 명령어 (1)

### ◎ ls

Windows의 "dir"과 같은 역할로, 해당 디렉터리에 있는 파일의 목록을 나열

예) # ls /etc/sysconfig

### ◎ cd

디렉터리를 이동

예) # cd ../etc/sysconfig

### ◎ pwd

현재 디렉터리의 전체 경로를 출력

'.' (현재 디렉터리)  
'..' (현재의 상위 디렉터리)

리눅스는 별도의 숨김 파일(Hidden File)이라는 속성이 존재하지 않는다.  
파일명이나 디렉터리의 제일 앞 글자를  
"."으로 하면 자동으로 숨김 파일이 된다.

옵션	설명
-a	<ul style="list-style-type: none"><li>경로 안의 모든 파일을 나열</li><li>‘.’으로 시작하는 파일들도 포함</li></ul>
-c	<ul style="list-style-type: none"><li>파일 최근 변경 시간에 따라 정렬하여 보여줌</li></ul>
-d	<ul style="list-style-type: none"><li>경로 안의 파일 목록을 나열하지 않고, 그 경로를 보여줌</li><li>- 이것은 쉘 스크립트에서 유용하게 쓰임</li></ul>
-f	<ul style="list-style-type: none"><li>경로 내용을 정렬하지 않음 : 이것은 디스크에 저장된 순으로 보여줌</li><li>-a와 -U 옵션과 같은 뜻이며, -l, -s, -t. 옵션과 반대의 뜻임</li></ul>
-i	<ul style="list-style-type: none"><li>파일 왼쪽에 inode번호를 보여줌</li></ul>
-l	<ul style="list-style-type: none"><li>파일의 모든 정보 출력</li></ul>

## 리눅스 기본 명령어 (2)



cp

파일이나 디렉터리를 복사

예) # cp abc.txt cba.txt



rm

파일이나 디렉터를 삭제

예) # rm -rf abc



mv

파일과 디렉터리의 이름을 변경하거나 위치 이동 시 사용

예) mv abc.txt www.txt



mkdir

새로운 디렉터를 생성

예) # mkdir abc

옵션	설명
-a	<ul style="list-style-type: none"><li>▪ 원본 파일의 속성, 링크 정보 등을 유지하며 복사</li><li>▪ -dpR 옵션과 같은 역할</li></ul>
-b	<ul style="list-style-type: none"><li>▪ 복사할 대상이 이미 있어 이것을 덮어쓰거나 지울 경우를 대비해 백업본을 만들</li></ul>
-f	<ul style="list-style-type: none"><li>▪ 만약 복사 대상 파일이 있는 경우 강제로 지우고 복사</li></ul>
-i	<ul style="list-style-type: none"><li>▪ 만약 복사 대상 파일이 있는 경우 어떻게 처리할지 프롬프트를 나타나게 함</li></ul>
-l	<ul style="list-style-type: none"><li>▪ 하드링크 형식으로 복사</li><li>▪ 경로 복사는 불가능</li></ul>
-r	<ul style="list-style-type: none"><li>▪ 일반 파일인 경우 그냥 복사</li><li>▪ 원본이 경로인 경우, 경로와 함께 안에 있는 모든 하위 경로의 파일들이 복사</li></ul>
-u	<ul style="list-style-type: none"><li>▪ 복사할 대상이 이미 있고 이 파일의 변경 날짜가 같거나, 더 최근의 것이면 복사하지 않음</li></ul>
-v	<ul style="list-style-type: none"><li>▪ 각 파일의 복사 상태를 자세히 나타냄</li></ul>

# rm

옵션	설명
-d	<ul style="list-style-type: none"><li>▪ 'rmdir' 명령 대신에 'unlink'와 함께 경로 삭제</li><li>▪ '-d' 옵션은 비우고자 하는 경로의 잔여파일 여부를 확인하지 않고 'unlink' 함</li><li>▪ 지우고자 하는 경로에 잔여 파일이 있는 경우 해당 파일에 대한 종속성 문제 발생 가능 (접근 불가능 현상, 미아 파일)</li><li>▪ '-d' 옵션 사용한 이후엔 fsck(8) 파일시스템 검사를 권장     'fsck' 명령 사용 시에는 주의를 요함</li><li>▪ 시스템 관리자만이 사용 가능</li></ul>
-f	<ul style="list-style-type: none"><li>▪ 지울 파일이 없는 경우 아무런 메시지를 보여주지 않고 넘어 감</li><li>▪ 쉘 스크립트 안에서 사용될 때 유용하게 사용됨</li></ul>
-i	<ul style="list-style-type: none"><li>▪ 각 파일을 하나씩 지울 것인지 일일이 사용자의 확인을 받음</li><li>▪ 'y', 'Y'를 눌러야만 파일이 삭제됨</li></ul>
-r	<ul style="list-style-type: none"><li>▪ 일반 파일인 경우 그냥 삭제됨</li><li>▪ 경로인 경우 해당 경로의 하위 경로 및 파일을 모두 삭제</li></ul>

## 리눅스 기본 명령어 (3)

### ◎ rmdir

디렉토리를 삭제. (단, 비어 있어야 함)

예) # rmdir abc

### ◎ cat

텍스트로 작성된 파일을 화면에 출력

예) # cat a.txt b.txt

### ◎ head, tail

텍스트로 작성된 파일의 앞 10행 또는 마지막 10행만 출력

예) # head anaconda-ks.cfg

### ◎ more

텍스트로 작성된 파일을 화면에 페이지 단위로 출력

예) # more anaconda-ks.cfg

## 리눅스 기본 명령어 (4)

### ◎ less

more와 용도가 비슷하지만 기능이 더 확장된 명령

예) # less anaconda-ks.cfg

### ◎ file

File이 어떤 종류의 파일인지를 표시

예) # file anaconda-ks.cfg

### ◎ clear

명령창을 깨끗하게 지워줌

예) # clear

## 실습

- ◎ man
- ◎ date
- ◎ who -mH
- ◎ cat [경로]파일
- ◎ touch
- ◎ head, tail
- ◎ less
- ◎ file
- ◎ clear

- ◎ ls
- ◎ cd
- ◎ pwd
- ◎ cp [경로]원본 [경로]목적지파일
- ◎ cp [경로]원본 [경로]디렉터리
- ◎ mv
- ◎ rm
- ◎ mkdir
- ◎ rmdir



## 사용자와 그룹(1)

- ◎ 리눅스는 다중 사용자 시스템(Multi-User System) 임
- ◎ 기본적으로 root라는 이름을 가진 슈퍼유저(Superuser)가 있으며, 모든 작업을 할 수 있는 권한이 있음
- ◎ 모든 사용자를 하나 이상의 그룹에 소속되어 있음
- ◎ 사용자는 /etc/passwd 파일에 정의되어 있음

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ cat /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin  
bin:x:2:2:bin:/bin:/usr/sbin/nologin  
sys:x:3:3:sys:/dev:/usr/sbin/nologin  
sync:x:4:65534:sync:/bin:/bin/sync  
gnats:x:7:7:Gnats Bug Reporting System (/usr/lib/bugreporting)/usr/sbin/nologin  
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin  
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin  
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin  
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin  
_apt:x:103:65534:/:nonexistent:/usr/sbin/nologin  
pi:x:1000:1000,,,:/home/pi:/bin/bash
```

- ◎ 각 행의 의미는 다음과 같음  
사용자 이름:암호:사용자 ID:사용자가 소속된 그룹 ID:전체 이름:홈 디렉터리:기본 셸

## 사용자와 그룹(2)

- ◎ 사용자의 비밀번호는 /etc/shadow 파일에 정의되어 있음
- ◎ 그룹은 /etc/group 파일에 정의되어 있음

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ cat /etc/group  
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:pi  
tty:x:5:  
  
kvm:x:106:  
render:x:107:  
crontab:x:108:  
netdev:x:109:pi  
pi:x:1000:  
messagebus:x:110:  
ssh:x:111:
```

- ◎ 각 행의 의미는 다음과 같음  
그룹명:비밀번호:그룹 id:그룹에 속한 사용자명

## 사용자와 그룹 관련 명령어(1)

- ◎ **useradd**  
새로운 사용자를 추가  
예) # useradd -m -s /bin/bash newuser
- ◎ **passwd**  
사용자의 비밀번호를 지정하거나 변경  
예) # passwd newuser
- ◎ **usermod**  
사용자의 속성을 변경  
예) # usermod -G root newuser
- ◎ **userdel**  
사용자를 삭제  
예) # userdel newuser

사용자 생성시 옵션

- D : default values
- G : 그룹 지정
- m : 홈 디렉터리 생성
- s : 셸지정
- p : 파스워드

**NOTE:** 현 시점에서는  
superuser만이 사용자나  
그룹을 관리할 수  
있습니다. sudo 필요

## adduser 예)

```
pi@osboxes:~$ sudo adduser testuser
Adding user `testuser' ...
Adding new group `testuser' (1001) ...
Adding new user `testuser' (1001) with group `testuser' ...
Creating home directory `/home/testuser' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
No password supplied
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for testuser
Enter the new value, or press ENTER for the default
    Full Name []: test user
    Room Number []: c2
    Work Phone []: 010-9999-999
    Home Phone []:
    Other []:
Is the information correct? [Y/n] Y
pi@osboxes:~$
```

lslogins -u

## 사용자와 그룹 관련 명령어(2)

◎ **change**  
사용자의 암호를 주기적으로 변경하도록 설정  
예) # change -m 2 newuser

◎ **groups**  
현재 사용자가 속한 그룹을 보여줌  
예) # groups

◎ **groupadd**  
새로운 그룹을 생성  
예) # groupadd newgroup

◎ **groupmod**  
그룹의 속성을 변경  
예) # groupmod -n newgroup mygroup

## 사용자와 그룹 관련 명령어(3)

- ◎ groupdel  
그룹을 삭제  
예) # groupdel newgroup
- ◎ gpasswd  
그룹의 암호를 설정하거나, 그룹의 관리를 수행  
예) # gpasswd newgroup

## 실습

- ◎ useradd
- ◎ passwd
- ◎ usermod
- ◎ userdel

- ◎ groups
- ◎ groupadd
- ◎ groupmod

# 파일과 디렉터리의 소유와 허가권 (1)

## ◎ 파일의 리스트와 파일 속성

pi@raspberrypi: ~							
File Edit Tabs Help							
pi@raspberrypi:~\$ ls -l							
total 1083784							
drwxr-xr-x	2	pi	pi	4096	Dec 7 14:21	06_opencv	
-rw-r--r--	1	pi	pi	1642090	Dec 13 14:17	2021-12-13-141721_1920x1080_scr0t.png	
drwxr-xr-x	2	pi	pi	4096	Dec 2 02:31	Bookshelf	
drwxr-xr-x	3	pi	pi	4096	Dec 7 14:21	Desktop	
drwxr-xr-x	2	pi	pi	4096	Dec 2 02:36	Documents	
drwxr-xr-x	2	pi	pi	4096	Dec 2 02:36	Downloads	

-	<u>rw-r--r--</u>	1	root	root	0	9월 16 13:50	sample.txt
---	------------------	---	------	------	---	-------------	------------

파일 유형

파일 허가권

링크 수

파일 소유자 이름

파일 소유 그룹이름

파일 크기(Byte)

마지막 변경 날짜/시간

파일 이름



## 파일과 디렉터리의 소유와 허가권 (2)

### ◎ 파일 유형

- 디렉터리일 경우에는 d, 일반적인 파일일 경우에는 -가 표시

### ◎ 파일 허가권(Permission)

- "rw-", "r--", "r--" 3개씩 끊어서 읽음 (r은 read, w는 write, x는 execute의 약자)
- 첫 번째 "rw-"는 소유자(User)의 파일접근 권한
- 두 번째의 "r--"는 그룹(Group)의 파일접근 권한
- 세 번째의 "r--"는 그 외의 사용자(Other)의 파일접근 권한
- 숫자로도 표시 가능 (8진수)

소유자(User)			그룹(Group)			그 외 사용자(Other)		
r	w	-	r	-	-	r	-	-
4	2	0	4	0	0	4	0	0
6			4			4		

## 특수권한

- ◎ Set UID: -rwsr-xr-x (4755)
- ◎ Set GID: -rwxr-sr-x (2755)
- ◎ Sticky bit: drwxr-xr-t (1755)

이름	비트	설명
SetUID	4	<ul style="list-style-type: none"> <li>■ 실행 시 그 파일의 소유자 권한을 획득하여 실행됨</li> <li>■ 누군가 파일을 실행할 경우 파일의 실제 소유권을 가진 사용자가 실행하는 것과 동일한 권한으로 실행</li> <li>■ 설정 시 소유권한 값의 실행(x)부분이 "s", "S"로 표시</li> </ul>
SetGID	2	<ul style="list-style-type: none"> <li>■ 실행 시 해당 파일의 소유그룹의 권한으로 실행되며, 설정되면 그룹 권한 값의 실행(x)부분이 "s", "S"로 표시</li> </ul>
Sticky bit	1	<ul style="list-style-type: none"> <li>■ 디렉터리에만 지정</li> <li>■ 디렉터리 내부의 파일들은 파일의 소유자에게만 쓰기권한 제공</li> <li>■ 파일의 소유자가 누구나 저장할 수 있도록 권한을 설정하여도 해당 파일의 소유자가 아니면 쓰기관련 작업은 불가능</li> <li>■ 설정 시 그 외에 해당하는 실행(x)부분이 "t", "T"로 나타남</li> <li>■ 유닉스/리눅스의 /tmp 디렉터리가 대표적인 예</li> </ul>

## 파일과 디렉터리의 소유와 허가권 (3)

### ◎ chmod 명령

- 파일 허가권 변경 명령어
- 예) # chmod 777 sample.txt

### ◎ 파일 소유권(Ownership)

- 파일을 소유한 사용자와 그룹을 의미

### ◎ chown/chgrp 명령

- 파일의 소유권을 바꾸는 명령어
- 예) # chown centos.centos sample.txt 또는  
# chown centos sample.txt 및 # chgrp centos sample.txt

## 파일과 디렉터리의 소유와 허가권 (4)

### ◎ chattr 명령

- 파일 속성을 변경
- chattr [-RV] [-v 버전] [모드]
- 기존의 파일속성에 w권한이 있으면 파일 삭제/변경가능
- 파일에 데이터만 추가 가능하게 하거나, 중요한 파일로서 소유자라고 하더라도 실제로 삭제하는 경우를 방지하고자 할 경우에 사용

### ◎ lsattr 명령

- 파일 속성을 출력
- lsattr [-Radv] 파일들

옵션	기능
R	디렉터리와 그 안의 내용에 대해 회귀적으로 속성값을 출력함
a	‘.’로 시작하는 파일을 포함한 디렉터리 내 모든 파일을 열거함
d	디렉터리를 보여줄 때 그 안에 든 파일을 보여주는 것이 아니라 일반 파일과 마찬가지로 보여줌
v	파일 버전을 출력함

## chattr 속성

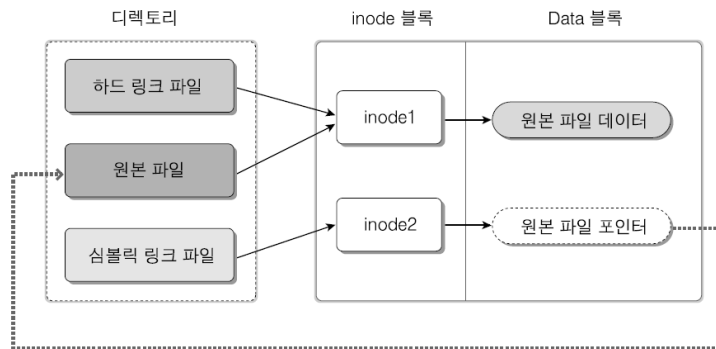
속성	설명
a	<ul style="list-style-type: none"><li>▪ append only로 파일 쓰기시에 오로지 추가모드로만 열 수 있음</li></ul>
c	<ul style="list-style-type: none"><li>▪ compressed 속성을 지닌 파일은 커널에 의해 디스크 상에 압축 상태로 저장</li><li>▪ 파일로부터 읽기 작업을 하면 압축이 풀린 자료가 반환</li><li>▪ 쓰기 작업은 디스크 상에 저장하기 전에 자동으로 압축</li></ul>
d	<ul style="list-style-type: none"><li>▪ dump(8) 명령 수행 시 백업되지 않음</li></ul>
i	<ul style="list-style-type: none"><li>▪ immutable</li><li>▪ 파일은 수정할 수 없음</li><li>▪ 지울 수도 이름을 변경할 수도 그리고 링크도 만들 수도 없으며 어떤 자료도 써질 수 없음</li><li>▪ 관리자만이 이 속성을 설정하거나 변경가능</li></ul>
s	<ul style="list-style-type: none"><li>▪ secure deletion</li><li>▪ 파일이 지워질 때는 일단 블록들이 모두 0으로 되어 파일이 안전하게 삭제가 됨</li></ul>
S	<ul style="list-style-type: none"><li>▪ Sync</li><li>▪ 파일 수정 시 변화가 디스크 상에 바로 동기화</li><li>▪ 'sync' 마운트 옵션을 몇몇 파일에게 부여한 것과 같음</li></ul>
u	<ul style="list-style-type: none"><li>▪ undelete</li><li>▪ 파일이 지워지면 그 내용이 저장됨</li><li>▪ 사용자는 다시 복구할 수 있음</li></ul>

## 실습

- ◎ chmod
- ◎ chown
- ◎ lsattr
- ◎ chattr

## 링크 (ln)

- ◎ 파일의 링크(Link)에는 하드 링크(Hard Link)와 심볼릭 링크(Symbolic Link 또는 Soft Link) 두 가지가 있음



심볼릭 링크는 Windows의 바로가기 아이콘과 개념이 비슷하다.

- ◎ 하드 링크를 생성하면 "하드링크파일"만 하나 생성되며 같은 inode1을 사용 (명령 : # ln 링크대상파일이름 링크파일이름)
- ◎ 심볼릭 링크를 생성하면 새로운 inode2를 만들고, 데이터는 원본 파일을 연결하는 효과 (명령 : # ln -s 링크대상파일이름 링크파일이름)

# In (링크)

- ◎ # ln 오리지널파일 하드링크
- ◎ # ln -s 오리지널파일 소프트링크
- ◎ 결과를 꼭 확인해 보세요 (# ls -la 또는 # stat [경로])

옵션	기능
-d, -F, --directory	<ul style="list-style-type: none"><li>▪ 경로의 하드링크를 허용함</li><li>▪ 이것은 시스템관리자만 가능함</li></ul>
-f, --force	<ul style="list-style-type: none"><li>▪ 대상 파일이 이미 있어도 그냥 지움</li></ul>
-s, --symbolic	<ul style="list-style-type: none"><li>▪ 심볼릭 링크</li><li>▪ 심볼릭 링크를 지원하지 않는 시스템에서 이 옵션을 사용할 경우에는 오류 메시지를 보여줌</li></ul>
-v, --verbose	<ul style="list-style-type: none"><li>▪ 각 파일의 작업 상태를 자세히 보여줌</li></ul>
-S, --suffix backup-suffix	<ul style="list-style-type: none"><li>▪ 만약에 대상이 이미 있어, 백업을 해야 할 경우에 그 백업 파일에서 사용할 파일 이름의 꼬리 문자를 지정함</li><li>▪ 이것은 이미 지정되어 있는 SIMPLE_BACKUP_SUFFIX 환경 변수를 무시하게 됨</li><li>▪ 만약 이 환경변수도 지정되어 있지 않고, 이 옵션도 사용하지 않는다면, 초기값으로 Emacs과 같이 '~' 문자를 사용함</li></ul>



## 실습

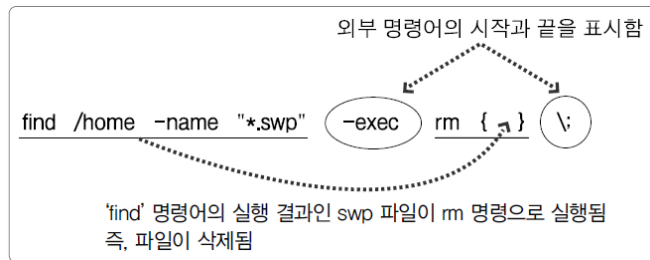
◎ ln

◎ ln -s

◎ stat

## 파일 위치 검색

- ◎ find [경로] [옵션] [조건] [action] : 기본 파일 찾기
  - [옵션] -name, -user(소유자), -newer(전,후), -perm(허가권), -size(크기)
  - [action] -print(디폴트), -exec (외부명령 실행)
  - 사용 예
    - # find /etc -name "\*.conf"
    - # find /bin -size +10k -size -100k
    - # find /home -name "\*.swp" -exec rm { } \;



- ◎ which 실행파일이름 : PATH에 설정된 디렉터리만 검색
- ◎ whereis 실행파일이름 : 실행 파일, 소스, man페이지 파일까지 검색
- ◎ locate 파일이름 : 파일 목록 데이터베이스에서 검색

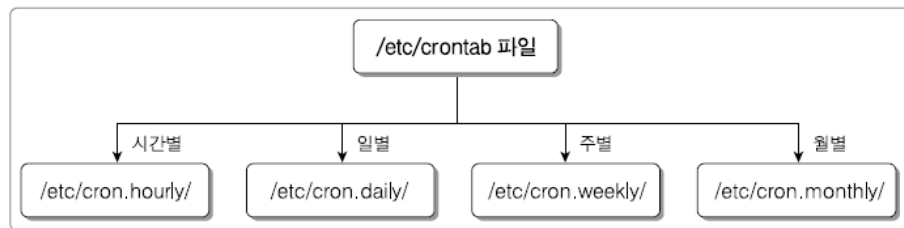
## 실습

- ◎ mlocate 설치 (# sudo apt install mlocate && sudo updatedb)
- ◎ locate
- ◎ which
- ◎ whereis
- ◎ find

# CRON (1)

## ◎ cron

- 주기적으로 반복되는 일을 자동적으로 실행될 수 있도록 설정
- 관련된 데몬(서비스)은 "crond", 관련 파일은 "/etc/crontab"



- /etc/crontab 예
  - 01 \* \* \* \* root run-parts /etc/cron.hourly
  - 02 4 \* \* \* root run-parts /etc/cron.daily
  - 03 4 \* \* 0 root run-parts /etc/cron.weekly
  - 42 4 1 \* \* root run-parts /etc/cron.monthly
- 첫 줄은 매시간 1분에 /etc/cron.hourly 디렉터리 안에 있는 명령들을 자동으로 실행한다

## CRON (2)

◎ cron table의 각 필드



◎ 각 필드

- \* : everytime | - : 범위 지정 | , : 구분 | \*/x : 매 x 분/시/일...

◎ crontab

- # crontab -l : cron table에 예약된 목록 출력
- # crontab -e : cron table을 편집
- # crontab -r : cron table에 예약된 목록 삭제

## 실습

◎ # touch echo.text

◎ # crontab -e

◎ 마지막줄에 아래를 삽입

```
*/* * * * * echo $PATH$ >> /home/pi/echo.text
```

◎ # tail -f echo.text

## 파일의 압축과 묶기 (1)

### ◎ 파일 압축

- 압축파일 확장명은 xz, bz2, gz, zip, Z 등
- xz나 bz2 압축률이 더 좋음

### ◎ 파일 압축 관련 명령

- xz : 확장명 xz로 압축을 하거나 풀어준다  
예) xz 파일명  
xz -d 파일명.xz
- bzip2 : 확장명 bz2로 압축을 하거나 풀어준다  
예) bzip2 파일명  
bzip2 -d 파일명.bz2
- bunzip2 : "bzip2 -d" 옵션과 동일한 명령어
- gzip : 확장명 gz으로 압축을 하거나 풀어준다  
예) gzip 파일명  
gzip -d 파일명.gz
- gunzip : "gzip -d" 옵션과 동일한 명령어

## 파일의 압축과 묶기 (2)

### ◎ 파일 묶기

- 리눅스(유닉스)에서는 '파일 압축'과 '파일 묶기'는 원칙적으로 별개의 프로그램으로 수행
- 파일 묶기의 명령어는 'tar'이며, 묶인 파일의 확장명도 'tar'이다

### ◎ 파일 묶기 명령(tar)

- tar : 확장명 tar로 묶음 파일을 만들어 주거나 묶음을 풀어 준다  
동작 : c(묶기), x(풀기), t(경로확인)  
옵션 : f(파일), v(과정보이기), J(tar+xz), z(tar+gzip), j(tar+bzip2)
- 사용 예  
# tar cvf my.tar /etc/sysconfig/ → 묶기  
# tar cvfJ my.tar.xz /etc/sysconfig/ /etc/sysconfig/ → 묶기 + xz 압축  
# tar xvf my.tar → tar 풀기  
# tar xvfJ my.tar.xz /etc/sysconfig/ → xz 압축 해제 + tar 풀기



## 네트워크 관련 필수 개념 (1)

- ◎ TCP/IP
  - 컴퓨터끼리 네트워크 상으로 의사소통을 하는 "프로토콜" 중 가장 널리 사용되는 프로토콜의 한 종류
- ◎ 호스트 이름(Hostname)과 도메인 이름(Domain name)
  - 호스트 이름은 각각의 컴퓨터에 지정된 이름
  - 도메인 이름(또는 도메인 주소)는 myhome.co.kr과 같은 형식
- ◎ IP 주소
  - 각 컴퓨터의 랜카드에 부여되는 중복되지 않는 유일한 주소
  - 4바이트로 이루어져 있으며, 각 자리는 0~255까지의 숫자
  - 예) Server의 IP 주소는 192.168.111.100
- ◎ 네트워크 주소
  - 같은 네트워크에 속해 있는 공통된 주소 (예 : 192.168.111.0)

# IP 주소 클래스

A	0	Network Address	Host Address
		0 ~ 127	0. 0. 0 ~ 255. 255. 255
B	10	Network Address	Host Address
		128. 0 ~ 191. 255	0. 0 ~ 255. 255
C	110	Network Address	Host Address
		192. 0. 0 ~ 223. 255. 255	0 ~ 255
D	11110	Multicast Address	
		224. 0. 0. 0 ~ 239. 255. 255. 255	
E	11111	Reserved	
		240. 0. 0. 0 ~ 255. 255. 255. 255	

일반 사용자

멀티캐스트용

향후 사용을  
위한 예약 주소

## 네트워크 관련 필수 개념 (2)

- ◎ 브로드캐스트(Broadcast) 주소
  - 내부 네트워크의 모든 컴퓨터가 듣게 되는 주소
  - 현재 주소의 제일 끝자리를 255로 바꾼 주소(C클래스)
- ◎ 게이트웨이(Gateway), 라우터(Router)
  - 라우터 = 게이트웨이
  - 네트워크 간에 데이터를 전송하는 컴퓨터 또는 장비
  - Vmware의 게이트웨이 주소는 192.168.111.2로 고정
- ◎ DNS(Domain Name System) 서버(= 네임 서버) 주소
  - URL을 해당 컴퓨터의 IP주소로 변환해 주는 서버
  - 설정 파일은 /etc/resolv.conf
  - Vmware를 사용하면 Vmware가 192.168.111.2번을 게이트웨이 및 DNS 서버로, 192.168.111.254를 DHCP 서버로 설정함.

## 중요한 네트워크 관련 명령어

- ◎ ifup <장치이름> 및 ifdown <장치이름>
  - 네트워크 장치를 On 또는 Off 시키는 명령어
- ◎ ifconfig <장치이름>
  - 장치의 IP주소 설정 정보를 출력
- ◎ Nslookup
  - DNS 서버의 작동을 테스트하는 명령어 (dnsutils)
- ◎ ping <IP주소 또는 URL>
  - 해당 컴퓨터가 네트워크상에서 응답하는지를 테스트하는 간편한 명령어

## 실습

- ◎ net-tools 설치 (sudo apt install net-tools dnsutils)
- ◎ ifconfig
- ◎ ping
- ◎ route
- ◎ nslookup

## 파이프, 필터, 리다이렉션

### ◎ 파이프(pipe)

- 두 개의 프로그램을 연결해 주는 연결통로의 의미
- "|"문자를 사용함
- 예) # ls -l /etc | more

### ◎ 필터(filter)

- 필요한 것만 걸러 주는 명령어
- grep, head, tail, wc, sort, [awk](#), [sed](#) 등
- 주로 파이프와 같이 사용
- 예) # tail -5 file.name

### ◎ 리다이렉션 (redirection)

- 표준 입출력의 방향을 바꿔 줌
- 예) ls -l > list.txt
- soft < list.txt > out.txt

# sort [option] [file]

옵션	설명
-b	▪ 필드나 키를 정렬할 때 라인 앞부분의 공백을 무시
-c	▪ 이미 정렬된 상태인지를 검사하고, 정렬된 상태라면 정렬하지 않음
-m	▪ 이미 정렬되어 있는 파일들을 합침 ▪ 이때 다시 정렬하는 것은 아님
-M	▪ (unknown) < 'JAN' < ... < 'DEC' 관계를 비교 ▪ -b 옵션을 내포
-k POS1[,POS2]	▪ POS1에서 키를 시작하고, POS2에서 끝냄
-n	▪ 문자열 숫자값에 따라 비교 ▪ -b 옵션을 내포
-o FILE	▪ 결과를 표준 출력이 아닌 지정된 FILE로 출력
-m	▪ 이미 정렬되어 있는 파일들을 합침 ▪ 이때 다시 정렬하는 것은 아님
-n	▪ 문자열 숫자값에 따라 비교 ▪ -b 옵션을 내포
-s	▪ 마지막 재정렬 비교를 하지 못하도록 정렬을 안정화시킴
-z	▪ 개행문자가 아닌 0 바이트로 행을 끝냄 ▪ find -print0 에서 사용가능

## 실습

- ◎ 파이프
- ◎ 리다이렉션
- ◎ grep
- ◎ sort

◎ 예)

```
ls /etc/ | sort | grep ^[d-e]
```

```
ls /etc/ | sort | grep ^[d-e] > list.txt
```



## 프로세스, 데몬 (1)

- ◎ 정의
  - 하드디스크에 저장된 실행코드(프로그램)가, 메모리에 로딩되어 활성화된 것
- ◎ 포그라운드 프로세스(**Foreground Process**)
  - 실행하면 화면에 나타나서 사용자와 상호작용을 하는 프로세스
  - 대부분의 응용프로그램
- ◎ 백그라운드 프로세스(**Background Process**)
  - 실행은 되었지만, 화면에는 나타나지 않고 실행되는 프로세스
  - 백신 프로그램, 서버 데몬 등
- ◎ 프로세스 번호
  - 각각의 프로세스에 할당된 고유번호
- ◎ 작업 번호
  - 현재 실행되고 있는 백그라운드 프로세스의 순차번호

## 프로세스, 데몬 (2)

### ◎ 부모 프로세스와 자식 프로세스

- 모든 프로세스는 부모 프로세스를 가지고 있음
- 부모 프로세스를 kill 하면, 자식 프로세스도 자동으로 kill 됨

### ◎ 프로세스 관련 명령

- **ps**
  - 현재 프로세스의 상태를 확인하는 명령어
  - "ps -ef | grep <프로세스 이름>"을 주로 사용함
- **kill**
  - 프로세스를 강제로 종료하는 명령어
  - "kill -9 <프로세스 번호>"는 강제 종료
- **pstree**
  - 부모 프로세스와 자식 프로세스의 관계를 트리 형태로 보여줌
- **nohup**
  - 사용자가 로그아웃하거나 터미널창을 닫아도 해당 프로세스가 백그라운드로 작업이 되어야 할 경우 사용

## ps

속성	기능
a	<ul style="list-style-type: none"> <li>모든 프로세스를 보여줌(보통 u, x 옵션과 연계해서 사용함)</li> </ul>
e	<ul style="list-style-type: none"> <li>해당 프로세스에 관련된 환경변수 정보를 함께 출력함</li> </ul>
f	<ul style="list-style-type: none"> <li>프로세스 간의 상속관계를 보여줌(PID, PPID 값이 표시)</li> </ul>
l	<ul style="list-style-type: none"> <li>프로세스의 정보를 옆으로 길게 보여줌</li> <li>우선순위값과 관련된 PRI와 NI 값을 확인할 수 있음</li> </ul>
u	<ul style="list-style-type: none"> <li>프로세스의 소유자에 대한 정보 등 매우 자세하게 보여줌</li> </ul>
x	<ul style="list-style-type: none"> <li>daemon process 등 터미널의 컨트롤과 관련이 없는 프로세스도 보여줌</li> <li>일반적으로 말하는 데몬 프로세스도 보여줌</li> <li>보통 ps 명령을 실행하면 현재 Shell의 자식 프로세스들만 보여주는데, 이 옵션으로 다른 프로세스들도 볼 수가 있음</li> <li>(BSD 계열) System V 계열인 경우에는 -x 대신에 -e를 사용할 수 있고, -u 대신에 -uf 옵션을 사용할 수 있음</li> </ul>

## ps tree

옵션	기능
a	각 프로세스의 명령행 인자까지 보여줌
H	현재 프로세스와 그것의 부모 프로세스를 하이라이트로 강조해서 보여줌
n	PID 값으로 정렬해서 보여주며, 같은 부모 프로세스를 가진 자식 프로세스들끼리 모아 출력함
p	PID 값을 같이 보여줌
l	특별한 긴 줄이 있어도 그대로 보여줌
u	UID 값을 같이 보여줌
G	프로세스 간의 관계를 보기 좋게 출력
c	독립적인 하위 트리를 보여주지 않으며, 같은 위치에 같은 프로세스가 있을 때는 합치지 않음

## 프로세스, 데몬 (3)

### ◎ top

- 현재 시스템에서 실행되고 있는 프로세스 상태를 실시간으로 화면에 출력
- # top

### ◎ jobs

- 백그라운드로 실행중인 프로세스나 현재 중지된 프로세스의 목록을 보여줌
- # jobs -lnp [작업스팩]

### ◎ fg [%작업번호]

- 백그라운드 프로세스를 포그라운드 프로세스로 전환

### ◎ bg

- 포그라운드 프로세스를 백그라운드 프로세스로 전환

## 실습

◎ top

◎ ps

◎ kill

◎ pstree

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with concentric rings, and the lines are thin and grey. The overall structure is organic and branching, resembling a molecular or biological network.

3.

# 셸 스크립트 프로그래밍

## bash 셸

◎ 기본 셸은 bash(Bourne Again SHell : '배시 셸')

◎ bash 셸의 특징

- Alias 기능(명령어 단축 기능)
- History 기능(위/아래 화살표키)
- 연산 기능
- Job Control 기능
- 자동 이름 완성 기능(탭키)
- 프롬프트 제어 기능
- 명령 편집 기능

◎ 셸의 명령문 처리 방법

- (프롬프트) 명령어 [옵션...] [인자...]
- 예) # rm -rf /mydir

## 환경 변수

- ◎ "echo \$환경변수이름" 으로 확인 가능
- ◎ "export 환경변수=값" 으로 환경 변수의 값을 변경
- ◎ 주요 환경변수

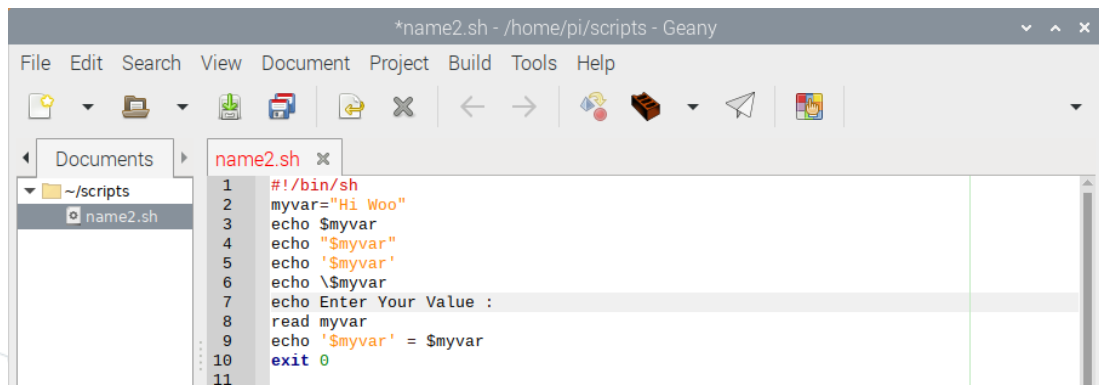
환경 변수	설명	환경 변수	설명
HOME	현재 사용자의 홈 디렉터리	PATH	실행 파일을 찾는 디렉터리 경로
LANG	기본 지원되는 언어	PWD	사용자의 현재 작업 디렉터리
TERM	로그인 터미널 타입	SHELL	로그인해서 사용하는 셸
USER	현재 사용자의 이름	DISPLAY	X 디스플레이 이름
COLUMNS	현재 터미널의 컬럼 수	LINES	현재 터미널 라인 수
PS1	1차 명령 프롬프트 변수	PS2	2차 명령 프롬프트(대개는 '>')
BASH	bash 셸의 경로	BASH_VERSION	bash 버전
HISTFILE	히스토리 파일의 경로	HISTSIZE	히스토리 파일에 저장되는 개수
HOSTNAME	호스트의 이름	USERNAME	현재 사용자 이름
LOGNAME	로그인 이름	LS_COLORS	'ls' 명령어의 확장자 색상 옵션
MAIL	메일을 보관하는 경로	OSTYPE	운영체제 타입



## 셸 스크립트 프로그래밍

- ◎ C언어와 유사하게 프로그래밍이 가능
- ◎ 변수, 반복문, 제어문 등의 사용이 가능
- ◎ 별도로 컴파일하지 않고 텍스트 파일 형태로 바로 실행
- ◎ vi나 gedit으로 작성이 가능
- ◎ 리눅스의 많은 부분이 셸 스크립트로 작성되어 있음

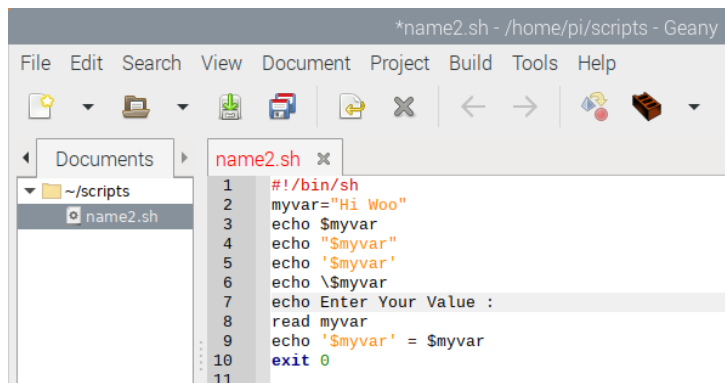
Text Editor나  
Geany로 작성가능



```
*name2.sh - /home/pi/scripts - Geany
File Edit Search View Document Project Build Tools Help
[Icons]
Documents
  ~scripts
    name2.sh
name2.sh x
1  #!/bin/sh
2  myvar="Hi Woo"
3  echo $myvar
4  echo "$myvar"
5  echo '$myvar'
6  echo \myvar
7  echo Enter Your Value :
8  read myvar
9  echo '$myvar' = $myvar
10 exit 0
11
```

## 셸 스크립트의 작성과 실행

### ◎ vi나 gedit으로 작성



```
*name2.sh - /home/pi/scripts - Geany
File Edit Search View Document Project Build Tools Help
Documents
~/scripts
name2.sh
1  #!/bin/sh
2  myvar="Hi Woo"
3  echo $myvar
4  echo "$myvar"
5  echo '$myvar'
6  echo \myvar
7  echo Enter Your Value :
8  read myvar
9  echo '$myvar' = $myvar
10 exit 0
11
```

셸 스크립트 파일의 확장명은 되도록 \*.sh로 주는 것이 좋다.

셸 스크립트 파일을 /usr/local/bin/ 디렉토리에 복사하고, 속성을 755로 변경해 주면 모든 사용자가 스크립트를 사용할 수 있다.  
이 작업은 보안상 root만 수행함

### ◎ 실행 방법

- ① "**sh <스크립트 파일>**"로 실행 또는
- ② "**chmod +x <스크립트 파일>**" 명령으로 실행 가능 속성으로 변경한 후에, "**./<스크립트파일>**"명령으로 실행

## 변수의 기본

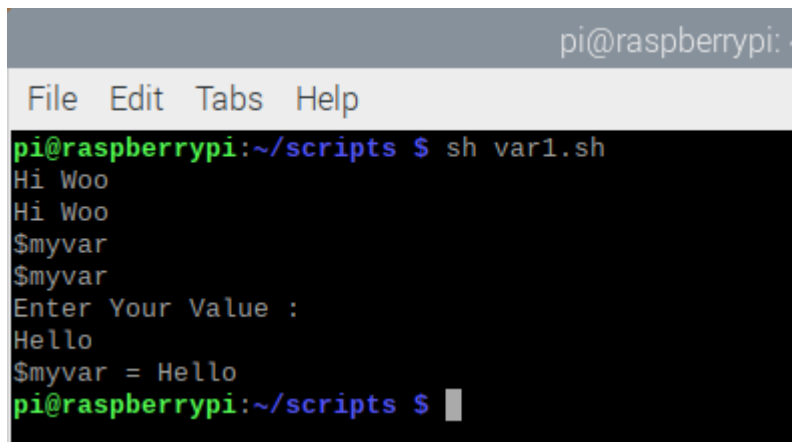
- ◎ 변수를 사용하기 전에 미리 선언하지 않으며, 변수에 처음 값이 할당되면서 자동으로 변수가 생성
- ◎ 모든 변수는 '문자열(String)'로 취급
- ◎ 변수 이름은 대소문자를 구분
- ◎ 변수를 대입할 때 '='좌우에는 공백이 없어야 함

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ testval = Hello  
bash: testval: command not found  
pi@raspberrypi:~ $ testval=Hello  
pi@raspberrypi:~ $ echo $testval  
Hello  
pi@raspberrypi:~ $ testval=Yes, Sir  
bash: Sir: command not found  
pi@raspberrypi:~ $ testval="Yes, Sir"  
pi@raspberrypi:~ $ echo $testval  
Yes, Sir  
pi@raspberrypi:~ $ testval=7+5  
pi@raspberrypi:~ $ echo $testval  
7+5  
pi@raspberrypi:~ $
```

## 변수의 입력과 출력

- ◎ '\$' 문자가 들어간 글자를 출력하려면 ' '로 묶어주거나 앞에 'w'를 붙임.
- ◎ " "로 변수를 묶어줘도 된다.

```
#!/bin/sh
myvar="Hi Woo"
echo $myvar
echo "$myvar"
echo '$myvar'
echo \ $myvar
echo Enter a value:
read myvar
echo '$myvar' = $myvar
exit 0
```

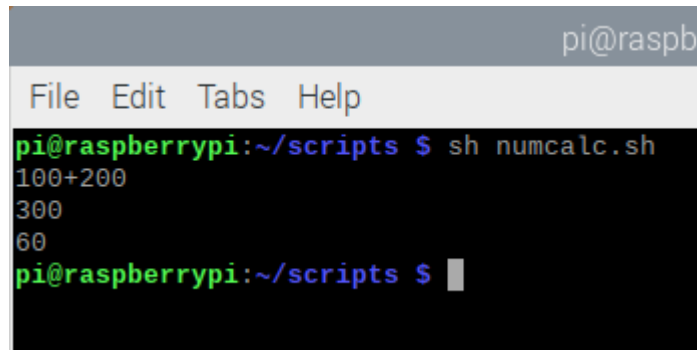
A terminal window on a Raspberry Pi. The title bar shows 'pi@raspberrypi: ~'. The menu bar has 'File Edit Tabs Help'. The prompt is 'pi@raspberrypi:~/scripts \$'. The user enters 'sh var1.sh'. The script outputs 'Hi Woo' twice, then '\$myvar' twice, then 'Enter Your Value :', then 'Hello', then '\$myvar = Hello'. The prompt returns to 'pi@raspberrypi:~/scripts \$' with a cursor.

```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~/scripts $ sh var1.sh
Hi Woo
Hi Woo
$myvar
$myvar
Enter Your Value :
Hello
$myvar = Hello
pi@raspberrypi:~/scripts $
```

## 숫자 계산

- ◎ 변수에 대입된 값은 모두 문자열로 취급
- ◎ 변수에 들어 있는 값을 숫자로 해서 +, -, \*, / 등의 연산을 하려면 `expr`를 사용
- ◎ 수식에 괄호 또는 곱하기(\*)는 그 앞에 꼭 역슬래시(\) 붙임

```
#!/bin/sh
num1=100
num2=$num1+200
echo $num2
num3=`expr $num1 + 200`
echo $num3
num4=`expr \( $num1 + 200 \) / 10 \* 2`
echo $num4
exit 0
```



```
pi@raspberrypi:~/scripts $ sh numcalc.sh
100+200
300
60
pi@raspberrypi:~/scripts $
```

## 파라미터(Parameter) 변수

◎ 파라미터 변수는 \$0, \$1, \$2...의 형태를 가짐

◎ 전체 파라미터는 \$\*로 표현

예)

명령어	yum	-y	install	gftp
파라미터 변수	\$0	\$1	\$2	\$3

```
#!/bin/sh
```

```
echo "Running file name is <$0>"
```

```
echo "First parameter is <$1>, the second parameter  
is <$2>"
```

```
echo "All parameters are <$*>"
```

```
exit 0
```

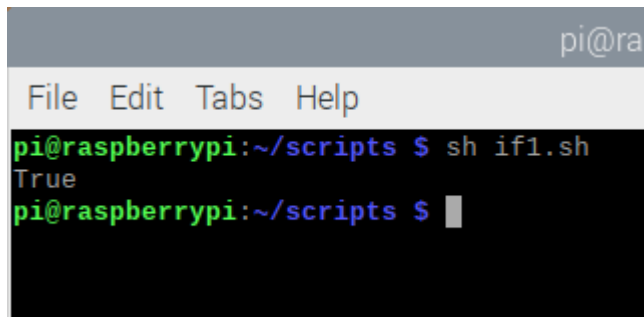
```
pi@raspberrypi:~/scripts $ sh paravar.sh value1 value2 value3  
Running file name is <paravar.sh>  
First parameter is <value1>, the second parameter is <value2>  
All parameters are <value1 value2 value3>
```

## 기본 if 문

◎ 형식  
if [ 조건 ]  
then  
    참일 경우 실행  
fi

"[ 조건 ]"의 사이의 각 단어에는  
모두 공백이 있어야 한다

```
#!/bin/sh
if [ "woo" = "woo" ]
then
    echo "True"
fi
exit 0
```



A terminal window on a Raspberry Pi. The prompt is 'pi@ras'. A menu bar shows 'File Edit Tabs Help'. The user enters 'sh if1.sh' and the output is 'True'. The prompt returns to 'pi@rasberrypi:~/scripts \$'.

```
pi@ras
File Edit Tabs Help
pi@rasberrypi:~/scripts $ sh if1.sh
True
pi@rasberrypi:~/scripts $
```

## if~else 문

◎ 형식  
if [ 조건 ]  
then  
    참일 경우 실행  
else  
    거짓인 경우 실행  
fi

중복 if 문을 위해서 else if가  
합쳐진 elif문도 사용할 수 있다.

```
#!/bin/sh
if [ "woo" != "woo" ]
then
    echo "True"
else
    echo "False"
fi
exit 0
```

```
pi@rasp
File Edit Tabs Help
pi@raspberrypi:~/scripts $ sh if2.sh
False
pi@raspberrypi:~/scripts $
```



## 조건문에 들어가는 비교 연산자

문자열 비교	결과
"문자열1" = "문자열2"	두 문자열이 같으면 참
"문자열1" != "문자열2"	두 문자열이 같지 않으면 참
-n "문자열"	문자열이 NULL(빈 문자열)이 아니면 참
-z "문자열"	문자열이 NULL(빈 문자열)이면 참

산술 비교	결과
수식1 -eq 수식2	두 수식(또는 변수)이 같으면 참
수식1 -ne 수식2	두 수식(또는 변수)이 같지 않으면 참
수식1 -gt 수식2	수식1이 크다면 참
수식1 -ge 수식2	수식1이 크거나 같으면 참
수식1 -lt 수식2	수식1이 작으면 참
수식1 -le 수식2	수식1이 작거나 같으면 참
!수식	수식이 거짓이라면 참

```
#!/bin/sh
if [ 100 -eq 200 ]
then
    echo "100 is the same as 200"
else
    echo "100 is not same as 200"
fi
exit 0
```

```
File Edit Tabs Help
pi@raspberrypi:~/scripts $ sh if3.sh
100 is not same as 200
pi@raspberrypi:~/scripts $
```

## 파일과 관련된 조건

```
#!/bin/sh
fname=/etc/hosts
if [ -f $fname ]
then
    head -5 $fname
else
    echo "There is no Web Service installed"
fi
exit 0
```

파일 조건	결과
-d 파일이름	파일이 디렉터리면 참
-e 파일이름	파일이 존재하면 참
-f 파일이름	파일이 일반 파일이면 참
-g 파일이름	파일에 set-group-id가 설정되면 참
-r 파일이름	파일이 읽기 가능이면 참
-s 파일이름	파일 크기가 0이 아니면 참
-u 파일이름	파일에 set-user-id가 설정되면 참
-w 파일이름	파일이 쓰기 가능 상태이면 참
-x 파일이름	파일이 실행 가능 상태이면 참

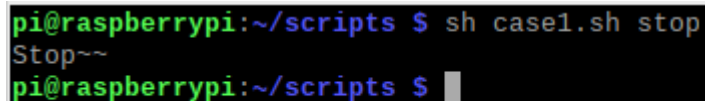
```
pi@raspberrypi:~/scripts $ sh if4.sh
127.0.0.1      localhost
::1           localhost ip6-localhost ip6-loopback
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters

pi@raspberrypi:~/scripts $
```

## case~esac 문 (1)

- ◎ if 문은 참과 거짓의 두 경우만 사용 (2중분기)
- ◎ 여러 가지 경우의 수가 있다면 case 문 (다중분기)

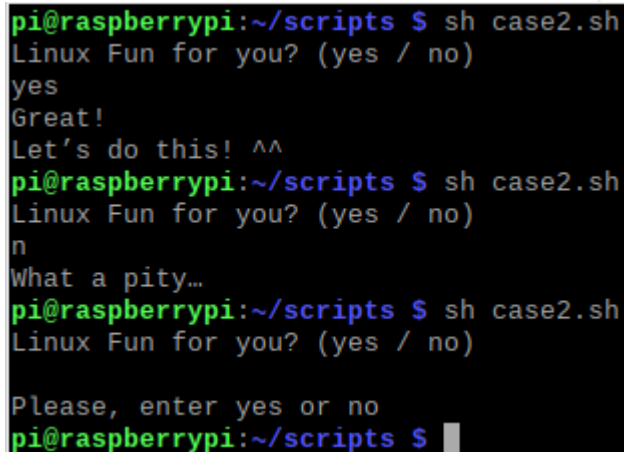
```
#!/bin/sh
case "$1" in
    start)
        echo "Start~~";;
    stop)
        echo "Stop~~";;
    restart)
        echo "Restart~~";;
    *)
        echo "What's happening???";;
esac
exit 0
```



```
pi@raspberrypi:~/scripts $ sh case1.sh stop
Stop~~
pi@raspberrypi:~/scripts $
```

## case~esac 문 (2)

```
#!/bin/sh
echo "Linux Fun for you? (yes / no)"
read answer
case $answer in
    yes | y | Y | Yes | YES)
        echo "Great!"
        echo "Let's do this! ^^";;
    [nN]*)
        echo "What a pity...";;
    *)
        echo "Please, enter yes or no"
        exit 1;;
esac
exit 0
```



```
pi@raspberrypi:~/scripts $ sh case2.sh
Linux Fun for you? (yes / no)
yes
Great!
Let's do this! ^^
pi@raspberrypi:~/scripts $ sh case2.sh
Linux Fun for you? (yes / no)
n
What a pity...
pi@raspberrypi:~/scripts $ sh case2.sh
Linux Fun for you? (yes / no)

Please, enter yes or no
pi@raspberrypi:~/scripts $
```

## AND, OR 관계 연산자

◎ and는 '-a' 또는 '&&'를 사용

◎ or는 '-o' 또는 '||'를 사용

```
#!/bin/sh
echo "Enter a file name to view"
read fname
if [ -f $fname ] && [ -s $fname ] ; then
    head -5 $fname
else
    echo "No file exist or size is 0"
fi
exit 0
```

```
pi@raspberrypi:~/scripts $ sh andor.sh
Enter a file name to view
../runthis_cv.py
#Import the Open-CV extra functionalities
import cv2

#This is to pull the information about what
classNames = []
pi@raspberrypi:~/scripts $ sh andor.sh
Enter a file name to view
../rubbish.rubbish
No file exist or size is 0
pi@raspberrypi:~/scripts $
```

## 반복문 – for문 (1)

◎ 형식  
for 변수 in 값1 값2 값3 ...  
do  
반복할 문장  
done

3행은  
for((i=1;i<=10;i++)) 또는 for i in 'seq 1 10'  
로 변경 할 수 있음

```
#!/bin/sh
hap=0
for i in 1 2 3 4 5 6 7 8 9 10
do
    hap=`expr $hap + $i`
done
echo "Total sum from 1 to 10: "$hap
exit 0
```

```
pi@raspberrypi:~/scripts $ sh for1.sh
Total sum from 1 to 10: 55
pi@raspberrypi:~/scripts $
```

## 반복문 – for문 (2)

◎ 현재 디렉터리에 있는 쉘 스크립트 파일(\*.sh)의  
파일명과 앞 3줄을 출력하는 프로그램

```
#!/bin/sh
for fname in $(ls *.sh)
do
    echo "-----$fname-----"
    head -3 $fname
done
exit 0
```

```
pi@raspberrypi:~/scripts $ sh for2.sh
-----andor.sh-----
#!/bin/sh
echo "Enter a file name to view"
read fname
-----case1.sh-----
#!/bin/sh
case "$1" in
    start)
        -----case2.sh-----
        #!/bin/sh
        echo "Linux Fun for you? (yes / no)"
        read answer
        -----for1.sh-----
        #!/bin/sh
        hap=0
        for i in 1 2 3 4 5 6 7 8 9 10
        -----for2.sh-----
        #!/bin/sh
        for fname in $(ls *.sh)
        do
            -----if1.sh-----
            #!/bin/sh
            if [ "woo" = "woo" ]
            then
                -----if2.sh-----
                #!/bin/sh
                if [ "woo" != "woo" ]
```

## 반복문 – while문 (1)

◎ 조건식이 참인 동안에 계속 반복

```
#!/bin/sh
while [ 1 ]
do
    echo "Raspbian Pi"
done
exit 0
```

[1] 또는 [:]가 오면 항상 참이 됨.  
그러므로 4행을 무한 루프로 반복함.  
Ctrl+C 로 루프 탈출

```
pi@raspberrypi:~/scripts $ sh while1.sh
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
Raspbian Pi
```



## 반복문 – while문 (2)

- ◎ 1에서 10까지의 합계를 출력 ('반복문 – for문 (1)' 슬라이드와 동일)

```
#!/bin/sh
hap=0
i=1
while [ $i -le 10 ]
do
    hap=`expr $hap + $i`
    i=`expr $i + 1`
done
echo "Total sum from 1 to 10: "$hap
exit 0
```

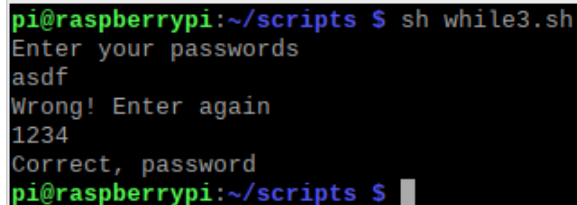
until 문은 조건식이 참일 때까지(= 거짓인 동안) 계속 반복  
4행을 until 문으로 바꾸면, until [ \$i -gt 10 ]

```
pi@raspberrypi:~/scripts $ sh while2.sh
Total sum from 1 to 10: 55
pi@raspberrypi:~/scripts $
```

## 반복문 – while문 (3)

◎ 비밀번호를 입력받고, 비밀번호가 맞을 때까지 계속 입력받는 스크립트

```
#!/bin/sh
echo "Enter your passwords"
read mypass
while [ $mypass != "1234" ]
do
    echo "Wrong! Enter again"
    read mypass
done
echo "Correct, password"
exit 0
```



```
pi@raspberrypi:~/scripts $ sh while3.sh
Enter your passwords
asdf
Wrong! Enter again
1234
Correct, password
pi@raspberrypi:~/scripts $
```

## until 문

- ◎ while문과 용도가 거의 같지만, until문은 조건식이 참일 때까지(=거짓인 동안) 계속 반복한다.
- ◎ while2.sh를 동일한 용도로 until문으로 바꾸려면 4행을 다음과 같이 바꾸면 된다.
  - `until [ $i -gt 10 ]`

## break, continue, exit, return 문

◎ break는 주로 반복문을 종료할 때 사용되며, continue는 반복문의 조건식으로 돌아가게 함. exit는 해당 프로그램을 완전히 종료함. Return은 함수 안에서 사용될 수 있으며 함수를 호출한 곳으로 돌아가게 함.

```
#!/bin/sh
echo "Infinite loop menu starts(b: break, c: continue, e: exit)"
while [ 1 ] ; do
read input
case $input in
    b | B )
        break ;;
    c | C )
        echo "You pressed continue to while loop"
        continue ;;
    e | E )
        echo "You pressed exit to terminate this program"
        exit 1 ;;
esac;
done
echo "You pressed break to exit while loop and display this message"
exit 0
```

## 사용자 정의 함수

◎ 형식  
함수이름 () { → 함수를 정의  
내용들...  
}  
함수이름 → 함수를 호출

```
#!/bin/sh
myFunction () {
    echo "Entered into a Function"
    return
}
echo "Starting a program"
myFunction
echo "Stopping a program"
exit 0
```

## 함수의 파라미터 사용

- ◎ 형식  
함수이름 () { → 함수를 정의  
\$1, \$2 ... 등을 사용  
}  
함수이름 파라미터1 파라미터2 ... → 함수를 호출

```
#!/bin/sh
hap () {
    echo `expr $1 + $2`
}
echo "10 plus 20 equals to..."
hap 10 20
exit 0
```

## eval

◎ 문자열을 명령문으로 인식하고 실행

```
#!/bin/sh  
str="ls -l anaconda-ks.cfg"  
echo $str  
eval $str  
exit 0
```

## export

◎ 외부 변수로 선언해 준다. 즉, 선언한 변수를 다른 프로그램에서도 사용할 수 있도록 해줌

◎ exp1.sh

```
#!/bin/sh  
echo $var1  
echo $var2  
exit 0
```

◎ exp2.sh

```
#!/bin/sh  
var1="Local variable"  
export var2="External variable"  
sh exp1.sh  
exit 0
```



## printf

- ◎ C언어의 printf() 함수와 비슷하게 형식을 지정해서 출력

```
#!/bin/sh
var1=100.5
var2="Linux is so fun!"
printf "%5.2f \n\n \t %s \n" $var1 "$var2"
exit
```

## set과 \$(명령어)

- ◎ 리눅스 명령어를 결과로 사용하기 위해서는 \$(명령어) 형식을 사용
- ◎ 결과를 파라미터로 사용하고자 할 때는 set과 함께 사용

```
#!/bin/sh
echo "Today's date is $(date) "
set $(date)
echo "Today is $4"
exit 0
```

## shift (1)

- ◎ 파라미터 변수를 왼쪽으로 한 단계씩 아래로 쉬프트시킴
- ◎ 10개가 넘는 파라미터 변수에 접근할 때 사용
- ◎ 단, \$0 파라미터 변수는 변경되지 않음

### ◎ 원하지 않는 결과의 소스

```
#!/bin/sh
myfunc () {
    echo $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11
}
myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
exit 0
```

## shift (2)

◎ shift 사용을 통한 앞 문제점을 해결

```
#!/bin/sh
myfunc() {
    str=""
    while [ "$1" != "" ] ; do
        str="$str $1"
        shift
    done
    echo $str
}
myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
exit 0
```



# Thanks!

## Any questions?

You can find me at:

@username & user@mail.me



**Free templates for all your presentation needs**



For PowerPoint and  
Google Slides



100% free for personal  
or commercial use



Ready to use,  
professional and  
customizable



Blow your audience  
away with attractive  
visuals