# 리눅스(쉘)

42경산 라피신 대비 사전 SW 준비과정

## 강사소개와수업계획

#### □ 홍 원 기 (대구대학교 컴퓨터정보공학부 정보보호전공 교수)

▶ 연구실: 정통2호관 7608호

▶ 전화: 053-850-6636

▶ 휴대폰: 010-4208-6636

➤ E-mail: <u>wkhong@daequ.ac.kr</u>



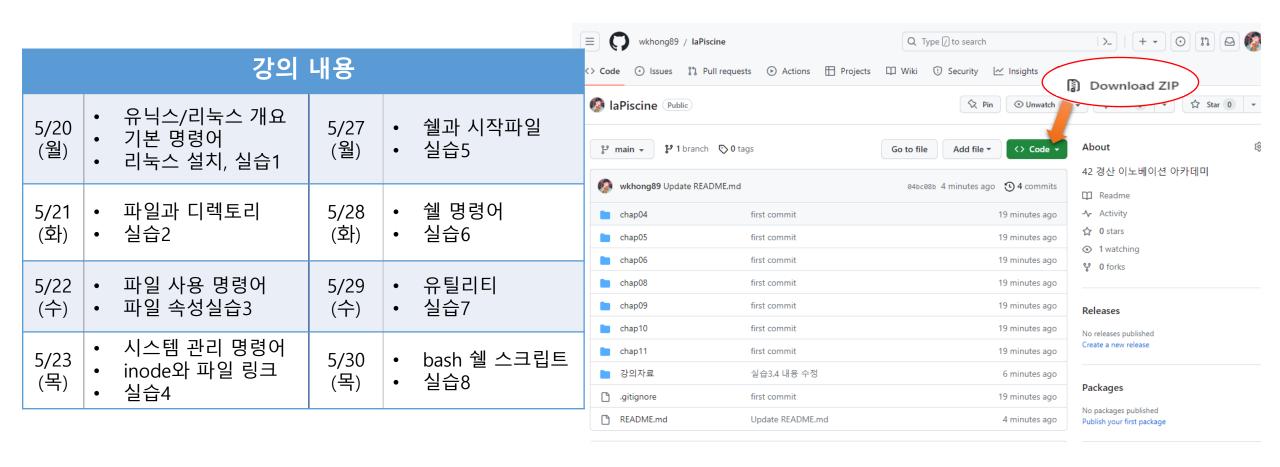
#### □ 수업 계획

▶ 이론과 실습 병행



#### 강의 및 실습 자료

• github.com/wkhong89/laPiscine에서 다운로드



01

유닉스/리눅스 개요

## 리눅스의 기원

#### □ 유닉스 (UNIX) 운영체제

- ▶ 1970년대 초에 AT&T 벨연구소에서 개발된 이후로 지속적으로 발전
- ▶ 스마트폰, PC, 서버 시스템, 슈퍼컴퓨터에까지 사용되고 있음
- ▶ 소프트웨어 경쟁력의 핵심

#### □ 유닉스/리눅스 기반 운영체제

- 1. 안드로이드(Android) OS
- 2. iOS
- 3. 맥(Mac) OS X
- 4. 리눅스(Linux)
- 5. BSD 유닉스(Unix)
- 6. 시스템 V
- 7. Sun 솔라리스(Solaris)
- 8. IBM AIX
- 9. HP HP-UX
- 10. Cray 유니코스(Unicos)



 $Ken\ Thompson\ (L)\ \ and\ Dennis\ R\ itchie\ (R)$ 

### 유닉스의설계철학

#### □ 단순성

- ➤ MIT MULTICS에 반대해서 최소한의 기능만 제공
- ▶ 자원에 대한 일관된 관점 제공

#### □이식성

- ▶ 이식성을 위해 C 언어로 작성
- ▶ 다양한 플랫폼에 이식 가능
- ▶ 스마트폰, PC, 서버, 슈퍼컴퓨터 등

#### □개방성

▶ 소스 코드 공개와 같은 개방성



그림 1.1 유닉스의 이식성

## 유닉스의특징

#### □ 다중 사용자, 다중 프로세스

- ▶ 여러 사용자가 동시에 사용 가능
- ▶ 여러 프로그램이 동시에 실행
- ▶ 관리자 슈퍼유저가 있음.

#### □ 쉘 프로그래밍

▶ 명령어나 유틸리티 등을 사용하여 작성한 프로그램

#### □ 훌륭한 네트워킹

- ▶ 유닉스에서부터 네트워킹이 시작
- > ftp, telnet, WWW, X-window 등

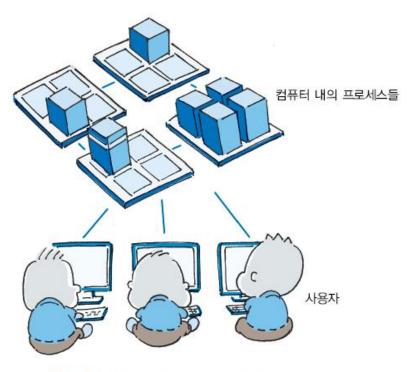


그림 1.2 다중 사용자 다중 프로세스

### 유닉스운영체제구조

#### 운영체제

컴퓨터의 하드웨어 자원을 운영 관리하고 프로그램을 실행할 수 있는 환경을 제공

#### □ 커널(kernel)

▶ 운영체제의 핵심으로 하드웨어 운영 및 관리

#### □ 시스템 호출(system call)

▶ 커널이 제공하는 서비스에 대한 프로그래밍 인터페이스 역할

#### □ 쉘(shell)

- ▶ 사용자와 운영체제 사이의 인터페이스
- ▶ 사용자로부터 명령어를 입력 받아 해석하여 수행해주는 명령어 해석기

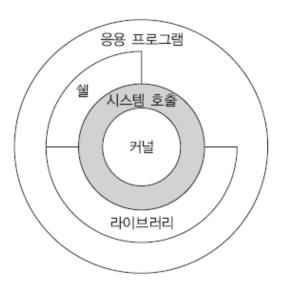
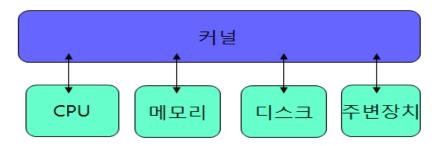


그림 1.3 유닉스 운영체제 구조

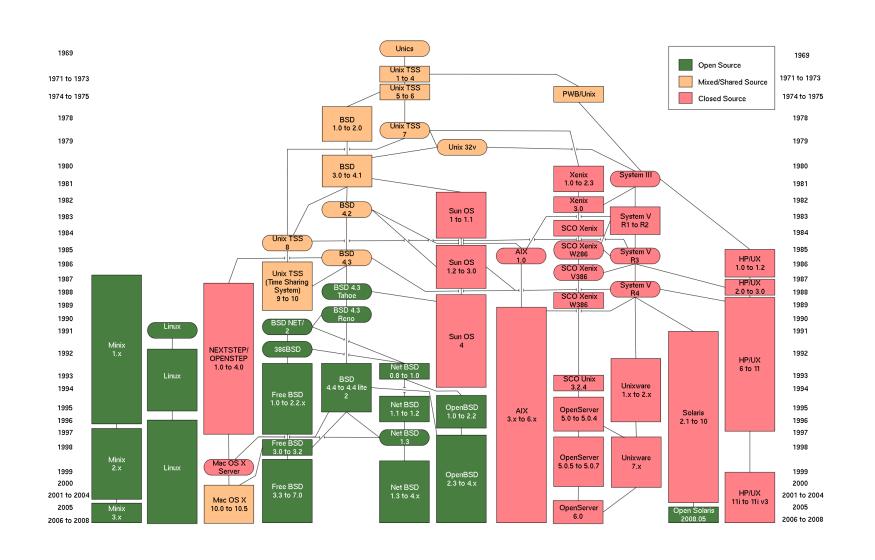
## 커널

#### □ 커널의 역할

- ▶ 하드웨어를 운영 관리하여
- ▶ 프로세스, 파일, 메모리, 통신, 주변장치 등을
- ▶ 관리하는 서비스를 제공한다.



## 유닉스 버전 트리[위키백과]



### GNU프로젝트



#### □ 공개 소프트웨어 프로젝트

- ➤ GPL에 따른 소프트웨어 배포
  - 누구나 자유롭게 "실행, 복사, 수정, 배포"
  - 누구도 그런 권리를 제한하면 안된다는 사용 허가권

#### □ 리처드 스톨만 (Richard Stallmann)

- > 자유 소프트웨어 재단 (FSF, Free Software Foundation) 설립자 (est. 1985)
  - GNU 프로젝트를 철학적, 법률적, 금융적으로 지원하기 위한 자선 단체
  - 목적
    - 소프트웨어의 본래 생산 유통 방식인 정보 공유 방식의 복원
    - 운영체제를 만들어 여러 사람들의 손을 거쳐 더 완성도 높은 소프트웨어를 만드는 것
    - 운영 체제만이 아닌 모든 소프트웨어를 자유 소프트웨어로 만드는 것
    - 소스코드 공개를 통해 누구나 소프트웨어를 수정할 수 있게 하는 것
    - 자유로운 복제와 배포를 허용하는 것



### Linux



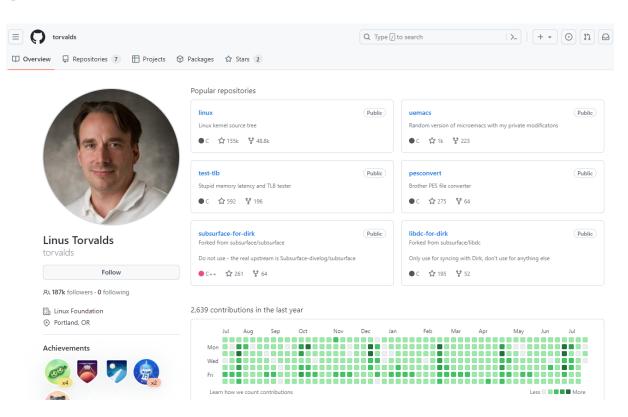
▶ 1991년 헬싱키 대학의 Linus Torvalds에 의해 개발됨

#### □ 소스코드 공개

- ▶ 인터넷 상에서 자원자들에 의해서 기능 추가 및 확장됨
- ➤ 공용 도메인 상의 무료 OS

#### □ 다양한 하드웨어 플랫폼에 포팅 가능

- ▶ PC, 워크스테이션, 서버, 메인프레임 등
- ▶ 놀라운 성능 및 안정성
- □ GNU 소프트웨어와 함께 배포
  - ➤ GNU/Linux 운영체제
  - ▶ 다양한 응용 프로그램



## 배포판 (distribution)

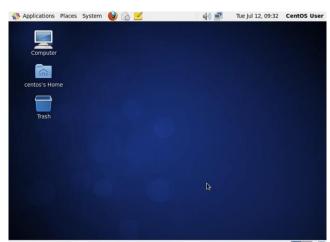
#### □ 배포판은 커널을 포함해 여러 외부 프로그램을 목적에 맞게 패키징한 것

- ▶ Linux kernel + shell (bash, zsh, tcsh) + GNU 유틸리티(ls, find, grep 등) + 개발도구 (gcc, gdb) + X Windows
- ▶ 현재 1000개 이상의 리눅스 배포판 존재

#### Linux Distribution Timeline

- ➤ Debian 계열
  - Debian, Ubuntu, Mint
- ➤ Redhat 계열
  - Redhat → RHEL (유료화), CentOS, Fedora, Oracle Linux































































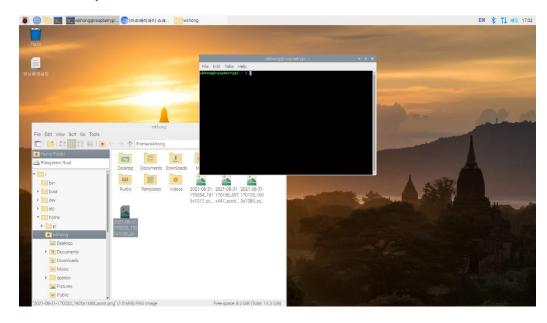
02 리눅스사용환경

## 직접로그인

#### □ 개인 유닉스/리눅스 시스템이 있는 경우

- ➤ X-윈도우(X-window)로 직접 로그인하여
- ▶ 바로 X-윈도우 시스템을 사용할 수 있다.

#### □ Rasberry-Pi OS





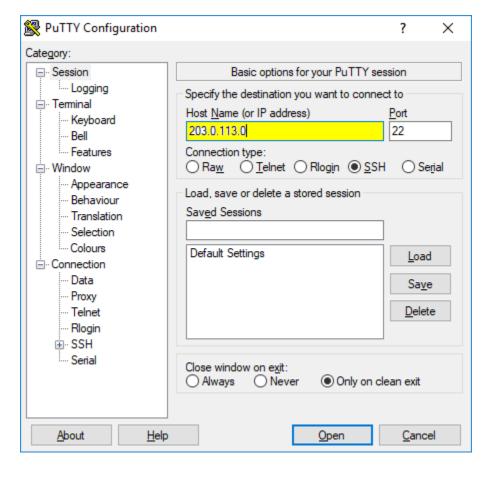


### 원격로그인

#### □ 회사/기관의 리눅스 서버에 원격 접속해서 사용

> ssh

➤ putty (Windows 프로그램)



## 가상머신

#### □ 가상 머신 (Virtual Machine)

- ▶ 컴퓨팅 환경을 소프트웨어로 구현한 것
- ▶ 컴퓨터시스템을 에뮬레이션한 소프트웨어
- ➤ 하이퍼바이저 (hypervisor)
  - 호스트컴퓨터에서 다수의 운영체제를 동시에 실행하기 위한 논리적 플랫폼
  - 가상머신을 생성하고 구동하는 소프트웨어

#### □ 가상 머신 소프트웨어

➤ Virtual Box, VMware, Parallels









## WSL (Windows Subsystem for Linux)

#### WSL

- ➤ Linux 용 Windows 하위 시스템
- ▶ 개발자가 기존 가상 머신의 오버헤드 또는 듀얼 부팅 설정 없이 Windows에서 Linux를 사용할 수 있게 지원
- ➤ WSL2: WSL의 새로운 버전
  - 파일 시스템 성능 개선
  - 전체 시스템 호출 호환성 개선

#### □ WSL 설치 가이드

## [실습1]

- □ 가상 환경에 설치된 우분투 리눅스 로그인하기
  - ➤ id: ubuntu
  - ➤ passwd: ubuntu
- □ 웹 브라우저 사용하기
- □ 설정 창 띄우기
- □ 텍스트 에디터 사용해보기
  - ▶ 한영 전환은?
- □ 파일 아이콘 클릭하기
- □ 터미널 창 띄우기

03 기본 명령어

### 기본명령어사용

#### □ 날짜 및 시간 확인

```
$ date
2016년 12월 26일 월요일 오후 01시 52분 02초
```

#### □ 시스템 정보 확인

- \$ hostname
  - 컴퓨터이름
  - DNS 주소
- \$ uname Linux
- \$ uname -a

Linux <hostname> 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86\_64 x86\_64 x86\_64 GNU/Linux

## 기본명령어사용

#### □ 사용자 정보 확인

```
$ whoami
wkhong
```

```
$ who
```

```
chang pts/1 2017-07-12 11:05 (:10.0)
brain pts/5 2017-07-12 13:46 (203.153.155.35)
```

. . .

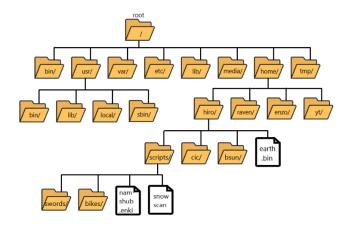
#### 디렉터리(Directory)

- 파일을 분류하기 위해 사용하는 이름공간
- 계층구조: 파일과 다른 하부 디렉토리로 구성
- 폴더(folder)라고도 함

□ 디렉토리 내용 확인

\$ 1s

Desktop Music Templates Documents Pictures Videos ...



## 기본명령어사용

#### □ 패스워드 변경

#### \$ passwd

Changing password for wkhong.

(current) UNIX password:

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

#### □ 화면 정리

\$ clear

### 온라인 매뉴얼: man

```
$ man ls
LS(1) User Commands LS(1)
NAME
ls - list directory contents
SYNOPSIS
ls [OPTION]... [FILE]...
DESCRIPTION
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
Mandatory arguments to long options are mandatory for short options too.
-a, --all
do not ignore entries starting with .
-A, --almost-all
do not list implied . and ..
Manual page ls(1) line 1 (press h for help or q to quit)
```

## 명령어에 대한 간단한 설명: whatis

#### \$ whatis 1s

```
ls (1) - 경로의 내용을 나열한다.
```

ls (1p) - list directory contents

04

파일과 디렉토리

### 파일의종류

#### □ 일반 파일(ordinary file)

- ▶ 데이터를 가지고 있으면서 디스크에 저장
- ▶ 텍스트 파일, 이진 파일

#### □ 디렉터리(directory) 또는 폴더(folder)

- ▶ 파일들을 계층적으로 조직화하는 데 사용되는 일종의 특수 파일
- ▶ 디렉터리 내에 파일이나 서브디렉토리들이 존재

#### □ 장치 파일(device special file)

- ▶ 물리적인 장치에 대한 내부적인 표현
- ▶ 키보드(stdin), 모니터(stdout), 프린터 등도 파일처럼 사용

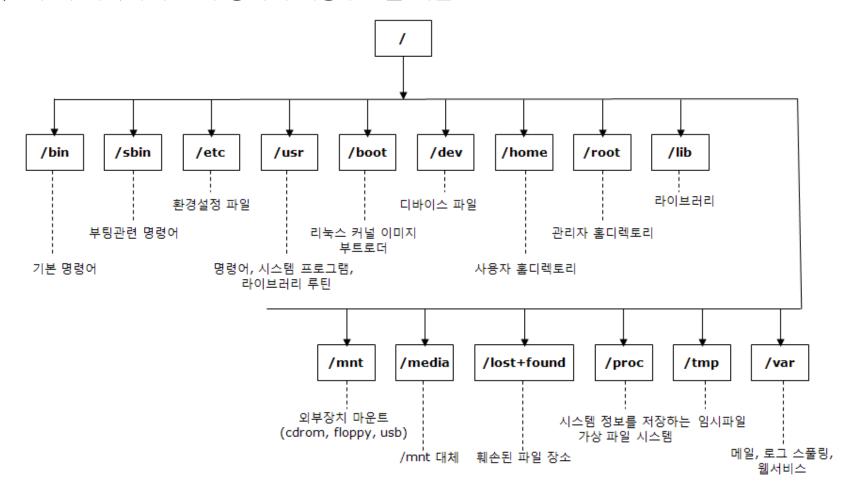
#### □ 심볼릭 링크 파일

▶ 어떤 파일을 가리키는 또 하나의 경로명을 저장하는 파일

## 디렉터리계층구조

#### □ 리눅스의 디렉터리

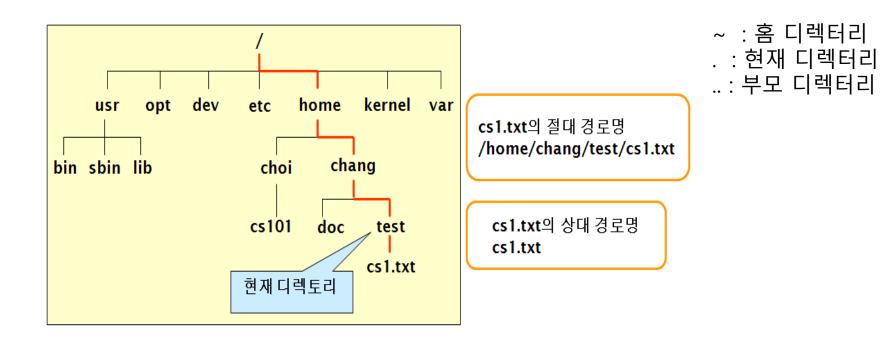
▶ 루트(/)로부터 시작하여 트리 형태의 계층구조를 이룸



## 홈디렉터리

#### □홈 디렉터리(home directory)

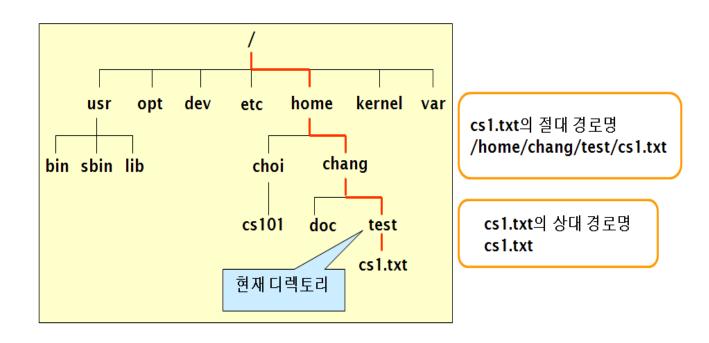
- ▶ 각 사용자마다 별도의 홈 디렉터리가 있음
- ▶ 사용자가 로그인하면 홈 디렉터리에서 작업을 시작함



29

## 경로명

- □ 파일이나 디렉터리에 대한 정확한 이름
- □ 절대 경로명(absolute pathname)
  - ▶ 루트 디렉터리로부터 시작하여 경로 이름을 정확하게 적는 것
- □ 상대 경로명(relative path name)
  - ▶ 현재 작업 디렉터리부터 시작해서 경로 이름을 적는 것



## 디렉터리이동: cd(change directory)

현재 작업 디렉터리를 지정된 디렉터리로 이동한다.

디렉터리를 지정하지 않으면 홈 디렉터리로 이동한다.

\$ cd [디렉터리]

### 현재작업디렉터리출력: pwd(printwarkingdreatory)

- □ 현재 작업 디렉터리(current working directory)
  - ▶ 현재 작업 중인 디렉터리
  - ▶ 로그인 하면 홈 디렉터리에서부터 작업이 시작된다.

```
$ pwd
현재 작업 디렉터리의 절대 경로명 출력
```

```
□ 예

$ pwd
    /home/wkhong/Desktop
$ cd ~
$ pwd
    /home/wkhong
```

## 명령어의 경로 확인: which

```
$ which 명령어
명령어의 절대경로를 보여줌
```

```
□ 여|

$ which ls

/bin/ls

$ which pwd

/usr/pwd

$ which passwd

/usr/passwd
```

### 디렉터리 리스트: ls(list)

```
$ ls(혹은 dir) [-asldFR] 디렉터리* 파일*
```

- 지정된 디렉터리의 내용을 리스트
- 디렉터리를 지정하지 않으면 현재 디렉터리 내용을 리스트
- 파일을 지정하면 해당 파일만을 리스트

```
■ 여

$ ls /

bin dev home lib64 mnt proc run srv tmp var

boot etc lib media opt root sbin sys usr

$ ls ~

Desktop Downloads Pictures Templates pl 다운로드

Documents Music Public Videos linux tmp 사진

$ cd Desktop

$ ls

cs1.txt
```

# ls 명령어 <del>옵</del>션

#### □ 주요 옵션

옵션	기능
-a	숨겨진 파일을 포함하여 모든 파일을 리스트
-S	파일의 크기를 K 바이트 단위로 출력
-	파일의 상세 정보를 출력
-d	디렉토리의 내용이 아닌 디렉토리 자신을 보여줌
-F	파일의 종류를 표시하여 출력
-R	모든 하위 디렉터리들을 리스트

## Is 명령어 옵션

- □ Is-s
  - ➤ -s(size) 옵션
  - ➤ 디렉터리 내에 있는 모든 파일의 크기를 K 바이트 단위로 출력
  - \$ 1s -s 총 4 4 cs1.txt
- □ Is -a
  - ➤ -a(all) 옵션
  - ▶ 숨겨진 파일들을 포함하여 모든 파일과 디렉터리를 리스트
  - ▶ "."은 현재 디렉터리, ".."은 부모 디렉터리
  - \$ 1s -a
    . . . cs1.txt

## Is 명령어 <del>옵</del>션

- □ |s -|
  - ➤ -l(long) 옵션
  - ➤ 파일 속성(file attribute) 출력
    - 파일 이름, 파일 종류, 접근권한, 소유자, 크기, 수정 시간 등

할당 크기 (allocation size)

- <u>4 -rw-r--r--</u> <u>1</u> <u>chang</u> <u>cs</u> <u>2088</u> <u>4월 16일 13:37</u> <u>cs1.txt</u>
- 1 2
- )

(5)

- 6
- 7

8

- 9
- ① 파일크기 ② 파일종류 ③ 접근권한 ④ 링크수 ⑤ 사용자 ID ⑥ 그룹 ID ⑦ 파일 크기
- ⑧ 최종 수정 시간 ⑨ 파일이름

실제 크기 (file size)

## Is 명령어 옵션

### □ Is -asl

```
$ 1s -as1
total 36
4 drwxr-xr-x 5 wkhong wkhong 4096 Sep 12 14:55 .
4 drwxr-xr-x 4 root root 4096 Sep 12 14:36 ...
4 -rw---- 1 wkhong wkhong 85 Sep 12 14:30 .bash_history
4 -rw-r--r-- 1 wkhong wkhong 220 Sep 12 14:11 .bash_logout
4 -rw-r--r-- 1 wkhong wkhong 3771 Sep 12 14:11 .bashrc
4 drwxr-xr-x 3 wkhong wkhong 4096 Sep 12 14:11 .cache
4 drwx---- 3 wkhong wkhong 4096 Sep 12 14:11 .config
4 -rw-r--r-- 1 wkhong wkhong 807 Sep 12 14:11 .profile
4 drwx----- 2 wkhong wkhong 4096 Sep 12 14:38 .ssh
```

## Is 명령어 옵션

# ls 명령어 <del>옵</del>션

- □ Is -R
  - ➤ -R(Recursive) 옵션
  - ▶ 모든 하위 디렉터리 내용을 리스트 한다.

### □ 예

```
$ 1s -R
$ 1s -R /
```

# 디렉터리 생성: mkdir(make directory)

### □사용법

```
$ mkdir [-p] 디렉터리+
디렉터리(들)을 새로 만듬
```

```
□ 예

$ cd ~ // 홈 디렉터리로 이동

$ mkdir test temp

$ ls -l

total 8

drwxr-xr-x 2 wkhong wkhong 4096 Sep 12 15:02 temp

drwxr-xr-x 2 wkhong wkhong 4096 Sep 12 15:02 test
```

## 디렉터리 생성: mkdir

- □ 옵션 -p
  - ▶ 필요한 경우에 중간 디렉터리를 자동으로 만들어 줌
- □ 예 : ~/dest 디렉터리가 없는 경우

```
$ mkdir ~/dest/dir1
mkdir: '/home/chang/dest/dir1' 디렉터리를 만들 수 없습니다: 그런 파일이나 디렉터리
가 없습니다
$ mkdir -p ~/dest/dir1
```

## 디렉터리 삭제 : rmdir(remove directory)

### □사용법

```
$ rmdir 디렉터리+
```

▶ 주의: 빈 디렉토리만 삭제할 수 있다.

```
S cd test
$ touch file
$ cd ..
$ rmdir test
rmdir: failed to remove 'test': 디렉터리가 비어있지 않음
$ cd test
$ rm file
$ cd ..
$ rmdir test
```

### [실습2]

- □ 현재 나의 디렉토리 확인하기
- □ 나의 홈디렉토리 아래에 laPiscine/day1 디렉토리 만들기
- □ 상대경로명을 이용해 day1 디렉토리로 이동하기
- □ day1의 파일 크기, 할당 크기, 생성일자, 파일종류, 소유자 확인하기
- □ /etc 디렉토리로 이동해서 passwd 파일의 크기, 파일 종류, 생성일자, 소유자 확인하기
- □ 명령어를 이용해서 Is 파일 위치 확인하고 Is가 있는 디렉토리로 이동해서 Is 파일 속성 확인하기
- □ /dev 디렉토리로 이동해서 파일들의 속성 확인하기
- □ 절대경로명을 이용해 나의 홈디렉토리 아래에 있는 day1 디렉토리로 이동하기
- □ laPaciine/day1 디렉토리 아래에 tmp 디렉토리를 만든 후 삭제하기
- □ 오늘 날짜 확인하기
- □ 리눅스 버전 확인하기
- □ 나의 id 확인하기

05 파일사용명령어

# 간단한 파일 만들기: cat

### □ cat 명령어 사용

```
$ cat > 파일
표준입력 내용을 모두 파일에 저장
파일이 없으면 새로 만듬
```

### □ 예

```
$ cat > cs1.txt
...
^D
```

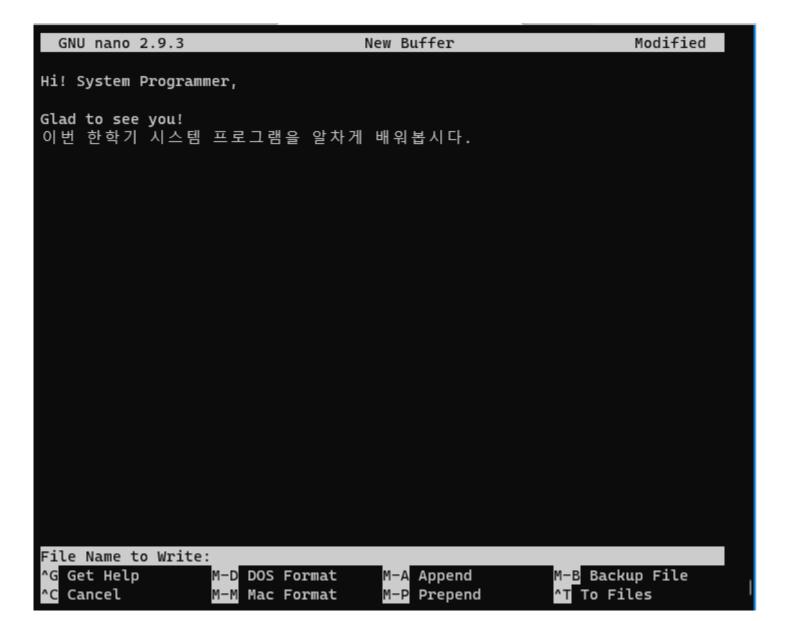
## 간단한 파일 만들기: touch

□ touch 명령어 사용

```
$ touch 파일
파일 크기가 0인 이름만 있는 빈 파일을 만들어 줌
```

```
이 $ touch cs1.txt $ ls -asl cs1.txt 0 -rw-rw-r--. 1 chang chang 0 5월 9 15:10 cs1.txt
```

# 편집기 프로그램: nano



# 파일내용출력

### □ 파일 내용 출력과 관련된 명령어들

- > cat, more, head, tail, wc 등
- \$ 명령어 파일
- \$ 명령어 파일\*
- \$ more 파일+

### 파일내용보기: cat

### □사용법

#### cat [-n] 파일\*

파일(들)의 내용을 그대로 화면에 출력

파일을 지정하지 않으면 표준입력 내용을 그대로 화면에 출력

### □ 예

#### \$ cat cs1.txt

Unix is a multitasking, multi-user computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna.

- - -

### 파일내용보기: cat

```
□ 예
  $ cat -n cs1.txt
  1 Unix is a multitasking, multi-user computer operating system originally
  2 developed in 1969 by a group of AT&T employees at Bell Labs, including
  3 Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy,
  4 and Joe Ossanna.
  . . .
  $ cat // 지정 파일 없음
  Hello World!
  Hello World!
  Bye!
  Bye!
  VD
```

### 페이지 단위로 파일 내용 보기: more

### □사용법

\$ more 파일+

파일(들)의 내용을 페이지 단위로 화면에 출력

### □ 예

#### \$ more cs1.txt

Unix is a multitasking, multi-user computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna.

. . .

During the late 1970s and early 1980s, the influence of Unix in academic circles led to large-scale adoption of Unix(particularly of the BSD variant, -- $\eta \Leftrightarrow --$ (59%)

## 파일 앞부분보기: head

### □사용법

#### \$ head [-n] 파일\*

파일(들)의 앞부분을 화면에 출력

파일을 지정하지 않으면 표준입력 내용을 대상으로 함

### □ 예

#### \$ head -5 cs1.txt

Unix is a multitasking, multi-user computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, and Joe Ossanna.

### 파일 뒷부분보기: tail

### □사용법

\$ tail [-*n*] 파일\*

파일(들)의 뒷부분을 화면에 출력

파일을 지정하지 않으면 표준입력 내용을 대상으로 함

### □ 예

#### \$ tail cs1.txt

Linux, which is used to power data centers, desktops, mobile phones, and embedded devices such as routers, set-top boxes or e-book readers. Today, in addition to certified Unix systems such as those already mentioned, Unix-like operating systems such as MINIX, Linux, Android, and BSD descendants (FreeBSD, NetBSD, OpenBSD, and DragonFly BSD) are commonly encountered.

The term traditional Unix may be used to describe a Unix or an operating system that has the characteristics of either Version 7 Unix or UNIX System V.

### 「단어세기: wc(word count)

### □사용법

```
$ wc [-lwc] 파일*
파일에 저장된 줄(l), 단어(w), 문자(c)의 개수를 세서 출력
파일을 지정하지 않으면 표준입력 내용을 대상으로 함
```

```
□ 여|

$ wc cs1.txt

38 318 2088 cs1.txt

$ wc -1 cs1.txt

38 cs1.txt

$ wc -w cs1.txt

318 cs1.txt

$ wc -c cs1.txt

2088 cs1.txt
```

### □사용법

```
$ cp [-i] 파일1 파일2
파일1을 파일2에 복사
-i는 대화형 옵션
```



```
s cp cs1.txt cs2.txt
$ ls -l cs1.txt cs2.txt
-rw-r--r-- 1 wkhong wkhong 0 Sep 12 15:18 cs1.txt
-rw-r--r-- 1 wkhong wkhong 0 Sep 12 15:18 cs2.txt
```

- □ 대화형 옵션: cp -i
  - ➤ 복사 대상 파일과 이름이 같은 파일이 이미 존재하면 덮어쓰기(overwrite)
  - ▶ 보다 안전한 사용법: 대화형 -i(interactive) 옵션을 사용

### □ 예

```
$ cp -i cs1.txt cs2.txt
cp: overwrite 'cs2.txt'? n
```

### □ 파일을 디렉터리로 복사

```
$ cp 파일 디렉터리
파일을 지정된 디렉터리에 복사
```

```
$ cp 파일1 ... 파일n 디렉터리
여러 개의 파일들을 지정된 디렉터리에 모두 복사
```

### □ 예

```
$ cp cs1.txt tmp/
$ ls -l tmp/cs1.txt
   -rw-r--r- 1 wkhong wkhong 0 Sep 12 15:20 temp/cs1.txt
$ cp cs1.txt cs2.txt tmp/
```

□ 디렉터리 전체 복사: cp-r

```
$ cp [-r] 디렉터리1 디렉터리2
r은 리커전 옵션으로 디렉터리1 전체를 디렉터리2에 복사
```

▶ 하위 디렉터리를 포함한 디렉터리 전체를 복사

```
□예
$ cp -r test temp
```

## | 파일이동: mv(move)

### □사용법

```
$ mv [-i] 파일1 파일2
파일1의 이름을 파일2로 변경
-i는 대화형 옵션
```



### □ 예

```
$ mv cs2.txt cs3.txt
$ ls -l
total 0
-rw-r--r- 1 wkhong wkhong 0 Sep 12 15:20 cs1.txt
-rw-r--r- 1 wkhong wkhong 0 Sep 12 15:21 cs3.txt
```

## 파일이동: mv(move)

- □ 대화형 옵션: cp -i
  - ▶ 이동 대상 파일과 이름이 같은 파일이 이미 존재하면 덮어쓰기(overwrite)
  - ▶ 보다 안전한 사용법: 대화형 -i(interactive) 옵션을 사용

### □ 예

```
$ mv -i cs1.txt cs3.txt
mv: overwrite 'cs3.txt'? n
```

## 파일이동: mv(move)

### □ 파일을 디렉터리로 이동

```
$ mv 파일 디렉터리
파일을 지정된 디렉터리로 이동
```

```
$ mv 파일1 ... 파일n 디렉터리
여러 개의 파일들을 지정된 디렉터리로 모두 이동
```

```
● 의 $ mv cs3.txt tmp/
$ ls -l tmp/cs3.txt
-rw-r--r-- 1 wkhong wkhong 0 Sep 12 15:21 cs3.txt
$ mv cs1.txt cs3.txt tmp/
```

# 파일이동: mv(move)

□ 디렉터리 이름 변경

```
$ mv 디렉터리1 디렉터리2
디렉터리1을 지정된 디렉터리2로 이름을 변경
```

```
□ 여
$ mkdir temp
$ mv temp tmp
```

# 파일삭제: rm(remove)

□사용법

```
$ rm [-i] 파일+
파일(들)을 삭제
-i는 대화형 옵션
```

```
□예

$ rm cs1.txt

$ rm cs1.txt cs3.txt

□대화형옵션:rm-i

$ rm -i cs1.txt

rm: remove 'cs1.txt'? n
```

### 디렉터리 전체 삭제

□ 디렉터리 전체 삭제: rm -r

```
$ rm [-ri] 디렉터리
-r은 리커전 옵션으로 디렉터리 아래의 모든 것을 삭제
-i는 대화형 옵션
```

#### □ 예

```
$ rm test
rm: cannot remove 'test': 디렉터리입니다
$ rmdir test
rmdir: failed to remove 'test': 디렉터리가 비어있지 않음
$ rm -ri test
rm: descend into directory 'test'? y
rm: remove regular file 'test/cs3.txt'? y
Rm: remove regular file 'test/cs1.txt'? y
rm: remove directory 'test'? y
```

06 파일속성

# 접근권한(permission mode)

### □ 파일에 대한 읽기(r), 쓰기(w), 실행(x) 권한

권한	파일	디렉터리
r	파일에 대한 읽기 권한	디렉터리 내에 있는 파일명을 읽을 수 있는 권한
W	파일에 대한 쓰기 권한	디렉터리 내에 파일을 생성하거나 삭제할 수 있는 권한
х	파일에 대한 실행 권한	디렉터리 내로 탐색을 위해 이동할 수 있는 권한

### □ 소유자(owner)/그룹(group)/기타(others)로 구분하여 관리

≽ भी: rwx r-x r-x



# 접근권한의예

접근권한	의미
rwxrwxrwx	소유자, 그룹, 기타 사용자 모두 읽기,쓰기,실행 가능
rwxr-xr-x	소유자만 읽기,쓰기,실행 가능, 그룹, 기타 사용자는 읽기,실행 가능
rw-rw-r	소유자와 그룹만 읽기,쓰기 가능, 기타 사용자는 읽기만 가능
rw-rr	소유자만 읽기,쓰기 가능, 그룹과 기타 사용자는 읽기만 가능
rw-r	소유자만 읽기,쓰기 가능 그룹은 읽기만 가능
rwx	소유자만 읽기,쓰기,실행 가능

## 초기 접근 권한과 umask

### □ 초기 접근 권한

- ▶ 파일 생성 시 기본으로 지정되는 기본 (default) 접근 권한을 말함
  - 디렉터리 기본 접근 권한: 777
  - 나머지 파일들의 기본 접근 권한: 666

#### umask

- ▶ 초기 접근 권한을 변경할 때 사용
- ➤ 파일 퍼미션 마스크(file permission mask)
- ➤ umask {3자리 8진수}
  - 기본 (default) 접근 권한 = 초기 접근 권한 umask
  - 예) umask 002
    - 파일 기본 접근 권한 = 666 002 = 664 (rw-rw-r--)
    - 디렉터리 기본 접근 권한 = 777 002 = 775 (rwxrwxr-x)
- ➤ 옵션 -S
  - umask 값을 문자열로 조회

# 접근권한 변경: chmod(change mode)

### □사용법

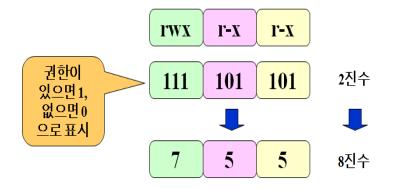
\$ chmod [-R] 접근권한 파일 혹은 디렉터리

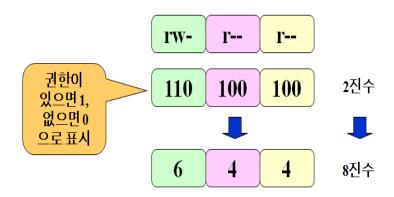
파일 혹은 디렉터리의 접근권한을 변경

-R 옵션을 사용하면 지정된 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 접근권한을 변경

## 접근권한 표현: 8진수

### □ 접근권한 8진수 변환





### • 사용 예

```
$ ls -l cs1.txt
-rw-r--r-- 1 wkhong .... cs1.txt
$ chmod 664 cs1.txt
$ ls -l cs1.txt
-rw-rw-r-- 1 wkhong ... cs1.txt
```

접근권한	진수
rwxrwxrwx	777
rwxr-xr-x	755
rw-rw-r	664
rw-rr	644
rw-r	640
rwx	700

# 접근권한 표현: 기호

### □ 기호를 이용한 접근권한 변경

사용자범위 연산자 권한

 $[u|g|o|a]^+$  [+|-|=]  $[r|w|x]^+$ 

구분	기호와 의미
사용자 범위	u(user:소유자), g(group:그룹), o(others:기타 사용자),
	a(all:모든 사용자)
연산자	+(권한 추가), -(권한 제거), =(권한 설정)
권한	r(읽기 권한), w(쓰기 권한), x(실행 권한)

## 기호를이용한접근권한변경

```
□ 예
  $ ls -1 cs1.txt
  -rw-r--r-- 1 wkhong wkhong 0 Sep 12 15:20 cs1.txt
  $ chmod g+w cs1.txt
  $ ls -1 cs1.txt
  -rw-rw-r-- 1 wkhong wkhong 0 Sep 12 15:20 cs1.txt
  $ chmod o-r cs1.txt
  $ ls -1 cs1.txt
  -rw-rw---- 1 wkhong wkhong 0 Sep 12 15:20 cs1.txt
  $ chmod g-w,o+rw cs1.txt
  $ ls -1 cs1.txt
  -rw-r--rw- 1 wkhong wkhong 0 Sep 12 15:20 cs1.txt
```

# 소유자 변경: chown(change owner)

#### □사용법

```
$ sudo chown 사용자 파일
$ sudo chown [-R] 사용자 디렉터리
관리자 권한으로 실행 (sudo)
파일 혹은 디렉터리의 소유자를 지정된 사용자로 변경한다.
-R 옵션: 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 소유자를 변경한다.
```

#### □ 예

```
$ chown guest1 cs1.txt
chown: changing ownership of 'cs1.txt': Operation not permitted
$ sudo chown guest1 cs1.txt
$ ls -l cs1.txt
-rw-r--rw- 1 guest1 wkhong 0 Sep 12 15:20 cs1.txt
$ echo "appending..." >> cs1.txt
-bash: cs1.txt: Permission denied
$ su guest1
$ echo "appending..." >> cs1.txt
```

# 그룹 변경: chgrp(change group)

### □사용법

\$ chgrp 그룹 파일

\$ chgrp [-R] 그룹 디렉터리

파일 혹은 디렉터리의 그룹을 지정된 그룹으로 변경

-R 옵션을 사용하면 지정된 디렉터리 아래의 모든 파일과 하위 디렉터리에 대해서도 그룹을 변경

## 최종수정시간변경: touch

□사용법

```
$ touch 파일
파일의 최종 사용 시간과 최종 수정 시간을 현재 시간으로 변경한다.
```

```
□예

$ touch cs1.txt

$ ls -l cs1.txt
```

# [실습 3] lapiciine/day2

- □ cat을 이용해 텍스트 파일 cat.txt 생성하기 (파일 내용: Wikipedia unix)
- □ nano, gedit 편집기를 이용해 텍스트 파일 nano.txt, gedit.txt 생성하기 (파일 내용: Wikipedia unix shell)
- □ cat, more, head, tail 을 이용해 파일 내용 출력
  - ▶ 라인 번호를 붙여 파일 내용 출력하기
  - ➤ more를 이용해 페이지 스크롤 up/down에 사용하는 키는? 종료키는?
  - ▶ 파일의 앞 부분 10줄만 출력하기
  - ▶ 파일의 뒷 부분 15줄만 출력하기

#### □ 복사

- ➤ cat.txt를 cat1.txt, cat2.txt로 복사하기
- > cat 디렉토리를 만들어 cat으로 시작하는 모든 텍스트 파일을 cat 디렉토리 아래로 복사하기
- ➤ nano 디렉토리를 만들어 nano.txt 파일을 nano 디렉토리 아래로 복사하기
- ➤ gedit 디렉토리를 만들어 gedit.txt 파일을 gedit 디렉토리 아래로 복사하기
- ▶ txt 디렉토리를 만들고 cat, nano, gedit 디렉토리와 아래 파일들을 모두 txt 디렉토리로 복사하기

# [실습 3] lapiscine/day2

#### □이동

- ➤ lapiscine/day2에 있는 cat2.txt를 cat3.txt로 이름 변경하기
- ▶ lapiscine/day2에 있는 cat으로 시작하는 모든 텍스트 파일을 cat\_mv 디렉토리 아래로 이동하기
- ➤ nano\_mv 디렉토리를 만들어 nano.txt 파일을 nano\_mv 디렉토리 아래로 이동하기
- ➤ gedit\_mv 디렉토리를 만들어 gedit.txt 파일을 gedit\_mv 디렉토리 아래로 이동하기
- ➤ txt\_mv 디렉토리를 만들고 cat\_mv, nano\_mv, gedit\_mv 디렉토리와 아래 파일들을 모두 txt\_mv 디렉토리로 이동하기

#### □ 삭제

▶ txt\_mv 디렉토리 아래에 있는 모든 파일 삭제하기

### □ 파일 접근권한 변경

- ➤ cat.txt 파일의 접근 권한 확인하기
- ▶ 그룹에 쓰기 권한 허용하기
- ▶ others에 읽기 권한 허용하지 않기

07

수퍼유저와 특권 명령어

## 시스템 관리자

### □ 슈퍼유저(superuser)

- ▶ 시스템을 관리하는 사용자
- ▶ 시스템의 모든 명령어와 모든 파일을 접근할 수 있는 권한을 가짐
- ➤ 슈퍼유저 ID: root

#### □ 슈퍼유저 로그인 방법

- ▶ 직접 root 계정으로 로그인
- ▶ 다른 계정으로 로그인 한 경우 : su (switch user) 명령어 사용\$ su root

\$ su

#### sudo

- ▶ 명령어를 관리자 모드로 실행하는 명령어
- ▶ 관리자 권한으로 접근할 수 있는 명령어나 파일 접근을 위해 수퍼유저 로그인이 불필요
- ► 사용법
  - \$ sudo [명령어]

# 시스템 관리자의 역할

- □ 사용자 계정 관리
  - ▶ 신규 사용자 추가/삭제
  - ▶ 사용자의 저장장치 쿼터 설정
- □ 사용자 그룹 관리
  - ▶ 그룹에 사용자의 추가/삭제
- □ 저장 장치 관리
- □ 네트워크 관리
- □ 소프트웨어 업그레이드

## 사용자계정생성/삭제명령어

#### 

Enter the new value, or press ENTER for the default
Full Name []: Guest1
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y

Changing the user information for guest1

## □ deluser -remove-home 사용자계정

➤ -remove-home 옵션: 홈 디렉토리 삭제

## 그룹생성/삭제 명령어

- □ 그룹(group): 여러 사용자들을 묶는 방법
  - ▶ 같은 그룹의 사용자는 그룹 파일에 대한 별도의 권한을 가질 수 있음
- □ addgroup 그룹계정
  - ▶ 그룹 생성 명령어
  - ▶ 관리자 권한으로 실행
  - \$ sudo addgroup usys Adding group `sp' (GID 1002) ... Done.
- □ delgroup 그룹계정
  - ▶ 그룹 삭제 명령어
  - ▶ 관리자 권한으로 실행
  - ▶ 그룹에 등록된 사용자가 없는 경우에만 삭제됨
  - \$ sudo delgroup sp1
  - Removing group `sp1' ...
  - Done.

## 그룹

### □ 그룹에 사용자 추가

➤ adduser [사용자명] [그룹명]

\$ sudo adduser wkhong sp

Adding user `wkhong' to group `sp' ...

Adding user wkhong to group sp

Done.

\$ sudo tail /etc/group

admin:x:113:

netdev:x:114:wkhong

wkhong:x:1000:

guest1:x:1001:

sp:x:1002:guest1,wkhong

guest2:x:1003:

## 사용자계정 편집 관리 명령어

#### □ usermod

▶ 사용자 id 변경

#### usermod - | <새로운계정 > -d <새로운 홈디렉토리 > -m <기존계정 >

- -1: 사용자 아이디 변경
- -d: 사용자 홈 디렉토리 변경
- -m: 홈 디렉토리 변경시 기존 파일 및 디렉토리를 옮겨줌; -d 옵션과 함께 사용
- ▶ 그룹 변경

usermod -q <변경그룹> <사용자계정>

▶ 그룹추가

#### usermod -G <변경그룹> <사용자계정>

- -a: 기존 2차 그룹이외에 추가로 2차 그룹 지정할 때 사용; -G옵션과 함께 사용
- ▶ 사용자 쉘, 계정 유효기간 변경

usermod -s <변경쉘> <사용자계정> usermod -e <변경날짜> <사용자계정>

# [실습4, 5]

- □ [리눅스(쉘)] 실습 1.pdf
- □ [리눅스(쉘)] 실습 2.pdf

08

inode와 파일 링크

## 아이노드 (inode)

#### inode

- ▶ 유닉스 계통 파일 시스템에서 파일에 관한 정보를 담고 있는 자료구조
- ▶ 파일 마다 하나의 inode가 할당됨
  - 저장장치가 담고있는 파일의 수 = inode의 수
- ➤ inode의 크기
  - 파일의 종류나 크기에 상관없이 고정된 크기



- > 메타 데이터
  - 파일 크기, 파일 생성 일자, 접근권한, 소유자 ID, 그룹 ID 등
  - 링크 수, 마지막 접근정보, 마지막 수정 정보, 아이노드 수정 정보
- ▶ 저장소(기억장치) 내 파일 위치

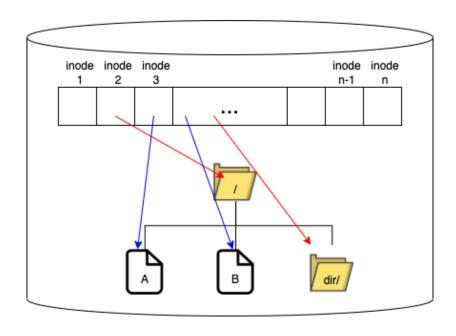
#### □ Is의 -i 옵션

➤ 파일의 inode 번호를 display

#### □ stat [file]

➤ file의 정보 display





# 링크

### □링크

▶ 기존 파일에 대한 또 하나의 새로운 이름

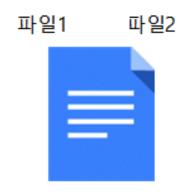
### □사용법

\$ In [-s] 파일1 파일2

파일1에 대한 새로운 이름(링크)로 파일2를 만들어 준다. -s 옵션은 심볼릭 링크

\$ In [-s] 파일1 디렉터리

파일1에 대한 링크를 지정된 디렉터리에 같은 이름으로 만들어 준다.



## 하드링크(hard link)

#### □ 하드 링크

- ▶ 기존 파일에 대한 새로운 이름
- ▶ 기존 파일의 inode를 공유

#### □ 예

```
$ In hello.txt hi.txt
```

**\$** 1s -1

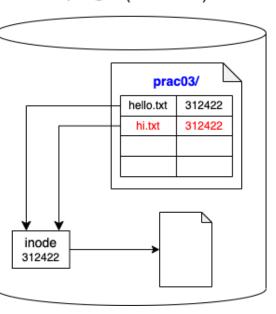
-rw----- 2 chang cs 15 11월 7일 15:31 hello.txt

-rw----- 2 chang cs 15 11월 7일 15:31 hi.txt

#### □질문

- ▶ 이 중에 한 파일의 내용을 수정하면 어떻게 될까?
- ▶ 이 둘 중에 한 파일을 삭제하면 어떻게 될까?

#### 하드 링크 (Hard Link)



# 심볼릭 링크(symbolic link)

#### 심볼릭 링크 (Symbolic Link)

### □ 심볼릭 링크

- ▶ 윈도우 시스템의 "바로가기 " 와 유사
- ➤ 새로운 inode 생성
  - 원본 파일의 inode 위치에 대한 정보를 가짐

```
prac03/
hello.txt 312422
hi.txt 312533
inode
312533
```

#### □ 예

```
$ ln -s hello.txt hi.txt
$ ls -l
-rw-r--r-- 1 wkhong wkhong 111 Sep 12 15:38 hello.txt
lrwxrwxrwx 1 wkhong wkhong 9 Sep 12 15:31 hi.txt -> hello.txt
$ ln -s /usr/bin/gcc cc
$ ls -l cc
lrwxrwxrwx 1 wkhong wkhong 12 Sep 12 15:42 cc -> /usr/bin/gcc
```

# [실습6]

□ [리눅스(쉘)] 실습 3.pdf

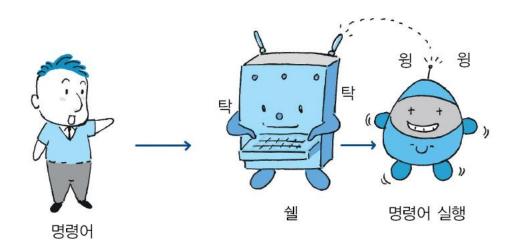
08

쉘

# 쉘(Shell)이란무엇인가?

### □ 쉘의 역할

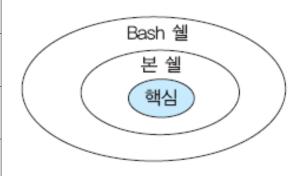
- ▶ 쉘은 사용자와 운영체제 사이에 창구 역할을 하는 소프트웨어
- ➤ 명령어 처리기(command processor)
- ▶ 사용자로부터 명령어를 입력 받아 처리



# 쉘의종류

## □ 유닉스/리눅스에서 사용 가능한 쉘의 종류

쉘의 종류	쉘 실행 파일	
본 쉘	/bin/sh	
콘 쉘	/bin/ksh	
C 쉘	/bin/csh	
Bash	/bin/bash	
tcsh	/bin/tcsh	





# 쉘의종류

#### □ 본 쉘(Bourne shell)

- ➤ 벨연구소의 스티븐 본(Stephen Bourne)에 의해 개발됨
- ▶ 유닉스에서 기본 쉘로 사용됨

#### □ 콘 쉘(Korn shell)

▶ 1980년대에는 역시 벨연구소에서 본 쉘을 확장해서 만듬.

#### ■ Bash(Bourne again shell)

- ➤ GNU에서 본 쉘을 확장하여 개발한 쉘
- ▶ 리눅스 및 맥 OS X에서 기본 쉘로 사용되면서 널리 보급됨
- ▶ Bash 명령어의 구문은 본 쉘 명령어 구문을 확장함

#### □ C 쉘(C shell)

- ➤ 버클리대학의 빌 조이(Bill Joy)
- ▶ 쉘의 핵심 기능 위에 C 언어의 특징을 많이 포함함
- ▶ BSD 계열의 유닉스에서 많이 사용됨
- ▶ 최근에 이를 개선한 tcsh이 개발됨어 되어 사용됨

# 로그인쉘(login shell)

- □ 로그인 하면 자동으로 실행되는 쉘
- □ 보통 시스템관리자가 계정을 만들 때 로그인 쉘 지정

/etc/passwd

root:x:0:0:root:/:/bin/bash

•••

wkhong:x:1000:1000:Won-Kee Hong:/home/wkhong:/bin/bash

# 로그인쉘변경

```
□ 쉘 변경
  $ csh
  %
  . . .
  % exit
□로그인 쉘 변경
  $ chsh
   Changing login shell for wkhong
   Old shell : /bin/sh
   New shell : /bin/csh
   $ logout
   login : wkhong
   passwd:
```

## 쉘의주요기능

### □ 명령어 처리

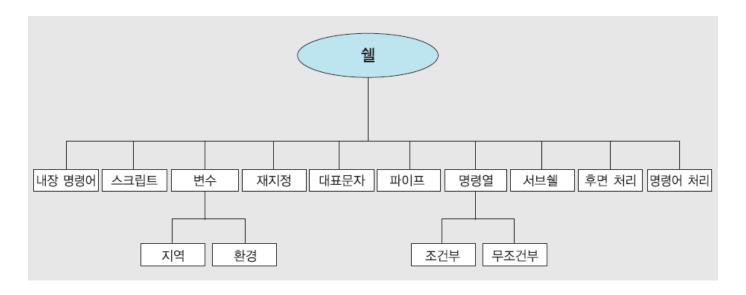
▶ 사용자가 입력한 명령을 해석하고 적절한 프로그램을 실행

#### □ 시작 파일

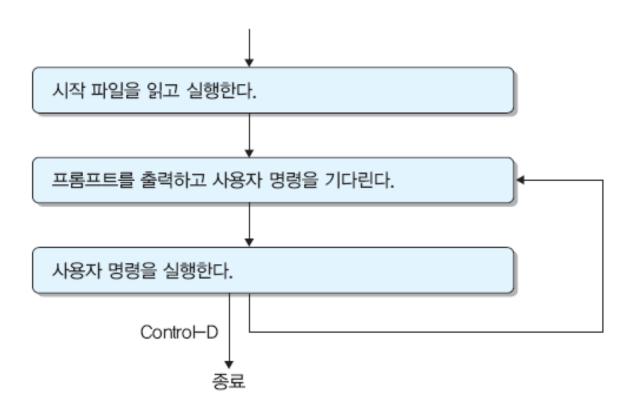
▶ 로그인할 때 실행되어 사용자별로 맞춤형 사용 환경 설정

### □ 스크립트

▶ 쉘 자체 내의 프로그래밍 기능



# 쉘의 실행 절차



# 쉘의 환경 변수

□ 환경변수 설정법

```
$ 환경변수명=문자열
```

환경변수의 값을 문자열로 설정

### □ 예

```
$ TERM=xterm
```

\$ echo \$TERM

xterm

# 쉘의 환경 변수

### □ 환경변수 보기

```
$ env
TERM=xterm
SHELL=/bin/sh
GROUP=cs
USER=chang
HOME=/home/chang
PATH=/usr/local/bin:/usr/bin: ...
```

### □ 사용자 정의 환경 변수

\$ MESSAGE=hello
\$ export MESSAGE

# 쉘의 시작 파일(start-up file)

#### □ 시작 파일

- ▶ 쉘마다 시작될 때 자동으로 실행되는 고유의 시작 파일
- ▶ 주로 사용자 환경을 설정하는 역할을 함
- ▶ 환경설정을 위해서 환경변수에 적절한 값 설정
- ① 시스템 시작 파일
  - 시스템의 모든 사용자에게 적용되는 공통적인 설정
  - 환경변수 설정, 명령어 경로 설정, 환영 메시지 출력, ...
- ② 사용자 시작 파일
  - 사용자 홈 디렉터리에 있으며 각 사용자에게 적용되는 설정
  - 환경변수 설정, 프롬프트 설정, 명령어 경로 설정, 명령어 이명 설정, ...

# 시작파일(start-up file)

쉘의 종류	시작파일 종류	시작파일 이름	실행 시기
본 쉘	시스템 시작파일	/etc/profile	로그인
	사용자 시작파일	~/.profile	로그인
Bash 쉘	시스템 시작파일	/etc/profile	로그인
	사용자 시작파일	~/.bash_profile	로그인
	사용자 시작파일	~/.bashrc	로그인, 서브쉘
	시스템 시작파일	/etc/bashrc	로그인
C 쉘	시스템 시작파일	/etc/.login	로그인
	사용자 시작파일	~/.login	로그인
	사용자 시작파일	~/.cshrc	로그인, 서브쉘
	사용자 시작파일	~/.logout	로그아웃

# 시작파일예

## □.profile

```
PATH=$PATH:/usr/local/bin:/etc
TERM=vt100
export PATH TERM
stty erase ^
```

### □ 시작 파일 바로 적용

\$ . .profile

## 전면 처리 vs 후면처리

## □ 전면 처리 (foreground processing)

▶ 입력된 명령어를 전면에서 실행하고 쉘은 명령어 실행이 끝날 때까지 대기\$ 명령어

### □ 후면 처리 (background processing)

▶ 명령어를 후면에서 실행하고 전면에서는 다른 작업을 실행하여 동시에 여러 작업 수행 가능\$ 명령어 &



# 후면처리예

```
$ (sleep 100; echo done) &
   [1] 8320

$ find . -name test.c -print &
   [2] 8325
```

## 후면작업확인

### □사용법

```
$ jobs [%작업번호]
후면에서 실행되고 있는 작업들을 리스트
작업 번호를 명시하면 해당 작업만 리스트
```

```
□여

$ jobs

[1] + Running ( sleep 100; echo done )

[2] - Running find . -name test.c -print

$ jobs %1

[1] + Running ( sleep 100; echo done )
```

# 후면작업을전면작업으로전환

## □사용법

```
$ fg %작업번호
작업번호에 해당하는 후면 작업을 전면 작업으로 전환
```

```
□여

$ (sleep 100; echo DONE) &

[1] 10067

$ fg %1

( sleep 100; echo DONE )
```

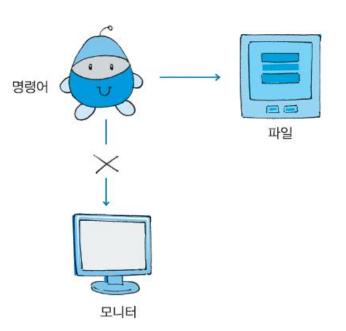
# 출력 재지정(output redirection)

## □사용법

```
$ 명령어 > 파일
명령어의 표준출력을 모니터 대신에 파일에 저장
```

```
□ 예
```

```
$ who > names.txt
$ cat names.txt
$ ls / > list.txt
$ cat list.txt
```



# 출력 재지정 이용: 간단한 파일 만들기

### □사용법

```
$ cat > 파일
표준입력 내용을 모두 파일에 저장
파일이 없으면 새로 생성
```

```
$ cat > list1.txt
Hi !
This is the first list.
^D

$ cat > list2.txt
Hello !
This is the second list.
^D
```

# 두개의파일을붙여서새로운파일만들기

• 사용법

```
$ cat 파일1 파일2 > 파일3
파일1과 파일2의 내용을 붙여서 새로운 파일3을 만듬
```

예

```
$ cat list1.txt list2.txt > list3.txt
$ cat list3.txt
Hi !
This is the first list.
Hello !
This is the second list.
```

# 출력추가

□ 사용법

```
$ 명령어 >> 파일
명령어의 표준출력을 모니터 대신에 파일에 추가
```

```
□ 예

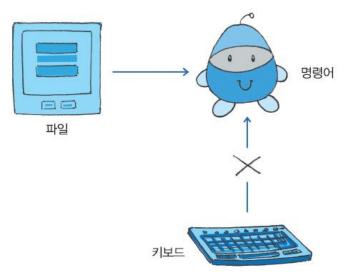
$ date >> list1.txt
$ cat list1.txt
Hi !
This is the first list.
Fri Sep 2 18:45:26 KST 2016
```

# 입력재지정(input redirection)

□ 사용법

```
$ 명령어 < 파일
명령어의 표준입력을 키보드 대신에 파일에서 받음
```

- □ 예 \$ wc < list1.txt 3 13 58 list1.txt
- □ 참고 \$ wc ... ^D \$ wc list1.txt



# 문서 내 입력(here document)

## □사용법

```
$ 명령어 << 단어
...
단어
명령어의 표준입력을 키보드 대신에 단어와 단어 사이의 입력 내용으로 받음
```

```
□여
$ wc << END
hello!
word count
END
2 4 20
```

# 오류재지정

#### □사용법

```
$ 명령어 2> 파일
명령어의 표준오류를 모니터 대신 파일에 저장
```

### □ 명령어의 실행결과

- ➤ 표준출력(standard output): 정상적인 실행의 출력
- ➤ 표준오류(standard error): 오류 메시지 출력

## □사용법

```
$ 1s -1 /bin/usr 2> err.txt
```

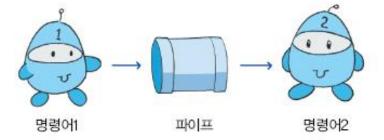
\$ cat err.txt

ls: cannot access /bin/usr: No such file or directory

# 파이프

#### □ 로그인 된 사용자들을 정렬해서 보여주기

```
$ who > names.txt
$ sort < names.txt</pre>
```



#### □사용법

\$ 명령어1 | 명령어2

명령어1의 표준출력이 파이프를 통해 명령어2의 표준입력이 된다.

□ 예

```
$\text{who} | sort agape pts/5 2월 20일 13:23 (203.252.201.55) chang pts/3 2월 20일 13:28 (221.139.179.42) hong pts/4 2월 20일 13:35 (203.252.201.51)
```

□ 예: 특정 디렉터리 내의 파일의 개수 출력 \$ 1s 디렉터리 | wc -w

# 명령어열(command sequence)

## □ 명령어 열

▶ 나열된 명령어들을 순차적으로 실행

## □사용법

```
$ 명령어1; ... ; 명령어n
```

```
$ date; pwd; ls
Fri Sep 2 18:08:25 KST 2016
/home/wkhong/linux/test
list1.txt list2.txt list3.txt
```

# 명령어그룹(command group)

#### □ 명령어 그룹

▶ 나열된 명령어들을 하나의 그룹으로 묶어 순차적으로 실행

### □사용법

```
$ (명령어1; ... ; 명령어n)
```

```
$ date; pwd; ls > out1.txt
Fri Sep 2 18:08:25 KST 2016
/home/wkhong/linux/test
$ (date; pwd; ls) > out2.txt
$ cat out2.txt
Fri Sep 2 18:08:25 KST 2016
/home/wkhong/linux/test
```

# 파일이름대치 (file name substitution)

## □ 대표문자를 이용한 파일 이름 대치

- ▶ 대표문자를 이용하여 한 번에 여러 파일들을 나타냄
- ▶ 명령어 실행 전에 대표문자가 나타내는 파일 이름들로 먼저 대치하고 실행

대표문자	의미
*	빈 스트링을 포함하여 임의의 스트링을 나타냄
?	임의의 한 문자를 나타냄
[]	대괄호 사이의 문자 중 하나를 나타내며 부분범위 사용 가능함.

```
$ gcc *.c
$ gcc a.c b.c test.c
$ ls *.txt
$ ls [ac]*
```

# 명령어 대치(command substitution)

## □ 명령어를 실행할 때 다른 명령어의 실행 결과를 이용

▶ `명령어` 부분은 그 명령어의 실행 결과로 대치된 후에 실행

```
$ echo 현재 시간은 `date`
현재 시간은 2023. 01. 01. (일) 12:00:00 KST
$ echo 현재 디렉터리 내의 파일의 개수 : `ls | wc -w`
현재 디렉터리 내의 파일의 개수 : 32
```

## 따옴표사용

## □ 따옴표를 이용하여 대치 기능을 제한

```
$ echo 3 * 4 = 12
3 cat.csh count.csh grade.csh invite.csh menu.csh test.sh = 12
$ echo "3 * 4 = 12"
3 * 4 = 12
$ echo '3 * 4 = 12'
3 * 4 = 12

$ name=나가수
$ echo '내 이름은 $name 현재 시간은 `date`'
내 이름은 $name 현재 시간은 `date`
$ echo "내 이름은 $name 현재 시간은 `date`"
내 이름은 나가수 현재 시간은 2016. 11. 11. (금) 10:27:48 KST
```

#### □정리

- ▶ 작은따옴표(')는 파일이름 대치, 변수 대치, 명령어 대치를 모두 제한
- ▶ 큰따옴표(")는 파일이름 대치만 제한
- ▶ 따옴표가 중첩되면 밖에 따옴표가 효력을 가짐

# [실습기

□ [리눅스(쉘)] 실습 4.pdf

# find 명령어

## □ find 명령어

▶ 파일 이름이나 속성을 이용하여 해당하는 파일을 찾음

## □사용법

\$ find 디렉터리 [-옵션]

옵션의 검색 조건에 따라 지정된 디렉터리 아래에서 해당되는 파일들을 모두 찾아 출력



# find 명령어

```
□ 예

$ find ~ -name src -print

/home/chang/linux/src

$ find ~ -name src -ls

89090 4 drwxrwxr-x 13 chang cs 4096 9월22 /home/chang/linux/src

$ find /usr -name *.c -print
```

# find 명령어: 검색 조건

검색 조건 및 처리 방법	설명
-name 파일명	파일명으로 찾는다.
-atime +n	접근 시간이 n일 이전인 파일을 찾는다.
-atime -n	접근 시간이 n일 이내인 파일을 찾는다.
-mtime +n	n일 이전에 수정된 파일을 찾는다.
-mtime -n	n일 이내에 수정된 파일을 찾는다.
-perm nnn	접근권한이 nnn인 파일을 찾는다.
-type x	파일 종류가 x인 파일들을 찾는다.
-size n	크기가 n 블록(512바이트)인 파일들을 찾는다.
-links n	링크 개수가 n인 파일들을 찾는다.
-user 사용자명	파일의 소유자가 사용자명인 파일을 찾는다.
-group 그룹명	그룹명을 갖는 그룹에 속한 파일을 찾는다.
-print	찾은 파일의 절대 경로명을 화면에 출력한다.
-ls	찾은 파일에 대해 1s -dils 명령어 실행 결과를 출력한다.
-exec cmd {};	찾은 파일들에 대해 cmd 명령어를 실행한다.

# find 명령어: 검색조건

□ 파일의 접근권한(-perm)으로 검색

```
$ find . -perm 700 -ls
```

## □ 파일의 접근 시간(-atime) 혹은 수정 시간(-mtime)으로 검색

+n: 현재 시각을 기준으로 n일 이상 전

n: 현재 시각을 기준으로 n일 전

-n: 현재 시각을 기준으로 n일 이내

```
$ find . -atime +30 -print
$ find . -mtime -7 -print
```

## find 명령어: 검색조건

## □ 파일의 소유자(-user)로 검색

```
$ find . -user wkhong -print
```

## □ 파일 크기(-size)로 검색

```
$ find . -size +1024 -print
```

## □ 파일 종류(-type)로 검색

```
d : 디렉터리 f: 일반 파일 1: 심볼릭 링크
```

b: 블록 장치 파일 c: 문자 장치 파일 s: 소켓 파일

\$ find ~ -type d -print

# find 명령어: 검색 조건 조합

□ 여러 검색 옵션을 조합해서 사용

```
□ 예

$ find . -type d -perm 700 -print

$ find . -name core -size +2048 -ls
```

# find 명령어: 검색된 파일 처리

- □ -exec 옵션
  - ▶ 검색한 모든 파일을 대상으로 동일한 작업(명령어)을 수행

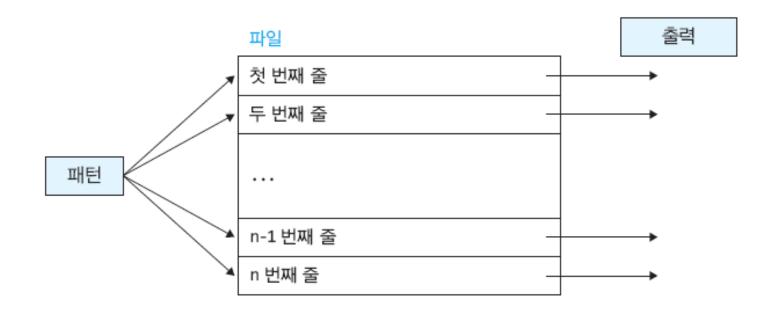
```
$ find . -name core -exec rm -i {} \;
$ find . -name *.c -atime +30 -exec ls -l {} \;
```

# grep 명령어

## □ 사용법

\$ grep 패턴 파일\*

파일(들)을 대상으로 지정된 패턴의 문자열을 검색하고, 해당 문자열을 포함하는 줄들을 출력



# grep 명령어

```
$ grep with you.txt
   Until you come and sit awhile with me
   There is no life - no life without its hunger;
   But when you come and I am filled with wonder,
$grep -w with you.txt
   Until you come and sit awhile with me
   But when you come and I am filled with wonder,
$grep -n with you.txt
   4:Until you come and sit awhile with me
   15:There is no life - no life without its hunger;
   17: But when you come and I am filled with wonder,
```

# grep 명령어의 옵션

옵션	기능	
-i	대소문자를 무시하고 검색한다.	
-	해당 패턴이 들어있는 파일 이름을 출력한다.	
-n	각 줄의 줄번호도 함께 출력한다.	
-V	명시된 패턴을 포함하지 않는 줄을 출력한다.	
-С	-c 패턴과 일치하는 줄 수를 출력한다.	
-W	패턴이 하나의 단어로 된 것만 검색한다.	

# grep 명령어

#### \$grep -i when you.txt

When I am down and, oh my soul, so weary
When troubles come and my heart burdened be
I am strong, when I am on your shoulders
But when you come and I am filled with wonder,

#### \$grep -v raise you.txt

When I am down and, oh my soul, so weary
When troubles come and my heart burdened be
Then, I am still and wait here in the silence
Until you come and sit awhile with me
I am strong, when I am on your shoulders
There is no life - no life without its hunger;
Each restless heart beats so imperfectly;
But when you come and I am filled with wonder,
Sometimes, I think I glimpse eternity

# grep: 정규 표현식의 활용

## □ 정규 표현식

▶ 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어

문자	의미	q
	임의의 한 문자	• 'ab'는 a로 시작해서 b로 끝나는 5글자 문자열
*	바로 앞의 것을 0번 이상의 반복	• 'a*b'는 b, ab, aab, aaab, 등의 문자열
[]	[과 ] 사이의 문자 중 하나를 의미 - 기호: 문자의 범위를 지정	<ul> <li>'[abc]d'는 ad, bd, cd를 의미</li> <li>[a-z]는 a부터 z까지 중 하나</li> </ul>
[^]	[^ 과 ] 사이의 문자를 제외한 나머 지 문자 중 하나	<ul> <li>'[^abc]d'는 ad, bd, cd는 포함하지 않고 ed, fd 등은 포함</li> <li>[^a-z]는 소문자가 아닌 모든 문자</li> </ul>
^, \$	각각 줄의 시작과 끝을 의미	<ul><li>'^문자열'은 문자열로 시작하는 줄을 의미</li><li>'문자열\$'은 문자열로 끝나는 줄을 의미</li></ul>

# 정규식사용예

#### □ \$ grep 'st..' you.txt

Then, I am still and wait here in the silence You raise me up, so I can stand on mountains You raise me up, to walk on stormy seas I am strong, when I am on your shoulders Each restless heart beats so imperfectly;

### □ \$ grep 'st.\*e' you.txt

Then, I am still and wait here in the silence You raise me up, to walk on stormy seas I am strong, when I am on your shoulders Each restless heart beats so imperfectly;

# □ \$ grep -w 'st.\*e' you.txt Then, I am still and wait here in the silence

# 파이프와함께 grep 명령어 사용

## □ 파이프와 함께 grep 명령어 사용

▶ 어떤 명령어를 실행하고 그 실행 결과 중에서 원하는 단어 혹은 문자열 패턴을 찾고자 할 때 사용

```
$ 1s -1 | grep chang
$ ps -ef | grep chang
```

# 파일 비교: diff

## □사용법

\$ diff [-i] 파일1 파일2

파일1과 파일2를 줄 단위로 비교하여 그 차이점을 출력

-i 옵션은 대소문자를 무시하여 비교

## □출력

▶ 첫 번째 파일을 두 번째 파일 내용과 같도록 바꿀 수 있는 편집 명령어 형태

# diff출력: 편집 명령어

## □ 추가(a)

▶ 첫 번째 파일의 줄 n1 이후에 두 번째 파일의 n3부터 n4까지의 줄들을 추가하면 두 파일은 서로 동일

```
n1 a n3,n4 > 추가할 두 번째 파일의 줄들
```

#### □ 예

```
$ diff you.txt me.txt
9a10,13
```

>

- > You raise me up, so I can stand on mountains
- > You raise me up, to walk on stormy seas
- > I am strong, when I am on your shoulders

# diff출력: 편집 명령어

### □ 삭제(d)

▶ 첫 번째 파일의 n1부터 n2까지의 줄들을 삭제하면 두 번째 파일의 줄 n3 이후와 서로 동일

```
n1,n2 d n3 < 삭제할 첫 번째 파일의 줄들
```

```
$ diff me.txt you.txt
10,13d9
<
    You raise me up, so I can stand on mountains
< You raise me up, to walk on stormy seas
< I am strong, when I am on your shoulders</pre>
```

# diff출력: 편집 명령어

## □ 변경(c)

▶ 첫 번째 파일의 n1부터 n2까지의 줄들을 두 번째 파일의 n3부터 n4까지의 줄들로 대치하면 두 파일은 서로 동일

```
n1,n2 c n3,n4
< 첫 번째 파일의 대치될 줄들
--
> 두 번째 파일의 대치할 줄들
```

```
$ diff 파일1 파일2
1 c 1
< This is the first file
--
> This is the second file.
```

# [실습8]

### □ find 명령어 실습

- ① find 명령어를 이용하여 /etc 디렉토리의 모든 파일 중 심볼릭 링크 파일들만을 상세하게 리스트하기
- ② find 명령어를 이용하여 /etc 디렉토리의 모든 파일 중 확장자가 .conf인 파일들의 상세 리스트를 conf.out 파일에 저장하기
- ③ Is, 파이프, grep 등을 사용하여 (2)와 같은 일을 하기
- ④ find 명령어를 사용하여 /usr 디렉토리의 모든 파일 중 접근 권한이 755인 것들을 모두 검색하기

## □ grep 명령어 실습

- ① grep 명령어를 사용하여 /etc/services 파일에서 tcp 서비스들을 찾아 그 개수를 출력하기
- ② grep 명령어를 사용하여 /etc/services 파일에서 tcp 서비스들을 찾아 이를 sort 명령어를 사용하여 정렬 하고 그 결과를 파일에 저장하기
- ③ grep 명령어를 사용하여 /etc/services 파일에서 #으로 시작하는 줄만 출력하기; #으로 시작하지 않는 줄들만 출력하기

10

bash 쉘 스크립트

# Bash(Borune-again shell)

- □ 리눅스, 맥 OS X 등의 운영 체제의 기본 쉘
- □ Bash 문법은 본 쉘의 문법을 대부분 수용하면서 확장
- □ 시작 파일(start-up file)
  - ▶ /etc/profile
    전체 사용자에게 적용되는 환경 설정, 시작 프로그램 지정
  - ➤ /etc/bashrc 전체 사용자에게 적용되는 별명과 함수들을 정의
  - ➤ ~/.bash\_profile 각 사용자를 위한 환경을 설정, 시작 프로그램 지정
  - ➤ ~/.bashrc 각 사용자를 위한 별명과 함수들을 정의

# Bash시작과정

```
/etc/profile
|
~/.bash_profile
|
~/.bashrc
|
로그인 쉘 프롬프트
```

# 시작 파일 예: .bash\_profile

```
# .bash_profile
# 사용자의 환경변수 설정 및 시작 프로그램
if [ -f ~/.bashrc ]
then
. ~/.bashrc
fi
PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
export USERNAME BASH_ENV PATH
```

# 시작 파일 예: .bashrc

```
# .bashrc
# 사용자의 별명 설정
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias ll='ls -al --color=yes'
# 시스템 시작 파일 실행
if [ -f /etc/bashrc ]
then
. /etc/bashrc
fi
```

# 별명

### □ alias 명령어

```
➤ 문자열이 나타내는 기존 명령에 대해 새로운 이름을 별명으로 정의

$ alias 이름=문자열

$ alias dir='ls -aF'

$ dir

$ alias h=history

$ alias 11='ls -1'
```

### □ 현재까지 정의된 별명들을 확인

```
$ alias # 별명 리스트
alias dir='ls -aF'
alias h=history
alias ll='ls -l'
```

### □ 이미 정의된 별명 해제

\$ unalias 단어

# 히스토리

- □ 입력된 명령들을 기억하는 기능
  - \$ history [-rh] [번호]
- □기억할 히스토리의 크기
  - \$ HISTSIZE=100
- □로그아웃 후에도 히스토리가 저장되도록 설정

```
$ HISTFILESIZE=100
```

```
$ history
```

- 1 ls
- 2 who
- 3 env
- 4 vi test.sh
- 5 chmod +x test.sh
- 6 test.sh
- 7 1s
- 8 date
- 9 history

. . .

# 재실행

형태	의미
!!	바로 전 명령 재실행
!n	이벤트 번호가 n인 명령 재실행
! 시작스트링	시작스트링으로 시작하는 최후 명령 재실행
!? 서브스트링	서브스트링을 포함하는 최후 명령 재실행

### □ 예

```
$ !! # 바로 전 명령 재실행
$ !20 # 20번 이벤트 재실행
$ !gcc # gcc로 시작하는 최근 명령 재실행
$ !?test.c # test.c를 포함하는 최근 명령 재실행
```

# 단순 변수(simple variable)

- □ 하나의 값(문자열)만을 저장할 수 있는 변수
  - \$ 변수이름=문자열
  - \$ city=seoul
- □ 변수의 값 사용

```
$ echo $city
seoul
```

- □ 변수에 어느 때나 필요하면 다른 값을 대입
  - \$ city=pusan
- □ 한 번에 여러 개의 변수를 생성
  - \$ country=korea city=seoul

# 단순 변수

### □ 한글 문자열을 값으로 사용

- \$ country=대한민국 city=서울
- \$ echo \$country \$city

대한민국 서울

### □ 따옴표를 이용하여 여러 단어로 구성된 문자열 저장 가능

\$ address="서울시 용산구"

# 리스트 변수(list variable)

□ 한 변수에 여러 개의 값(문자열)을 저장할 수 있는 변수

\$ 이름=( 문자열리스트 )

\$ cities=(서울 부산 목포)

### □ 리스트 변수 사용

리스트 사용	의미
\${name[i]}	리스트 변수 name의 i번째 원소
\${name[*]}	리스트 변수 name의 모든 원소
\${name[@]}	니므트 한구 Name의 또는 편요
\${#name[*]}	리스트 변수 name 내의 원소 개수
\${#name[@]}	니ㅡㅡ 근무 name 네ㅋ 건工 개구

# 리스트변수사용예

```
□ 리스트 변수 사용
   $ echo ${cities[*]}
   서울 부산 목포
   $ echo ${cities[1]}
   부산
□ 리스트의 크기
   $ echo ${#cities[*]} # 리스트 크기
   $ echo ${cities[3]}
□ 리스트 변수에 새로운 도시 추가
   $ cities[3]=제주
   $ echo ${cities[3]}
   제주
```

# 표준입력 읽기

#### □ read 명령어

- ▶ 표준입력에서 한 줄을 읽어서 단어들을 변수들에 순서대로 저장
- ▶ 마지막 변수에 남은 단어들 모두 저장

```
$ read 변수1 ... 변수n

$ read x y
Merry Christmas!
$ echo $x
Merry
$ echo $y
Christmas!

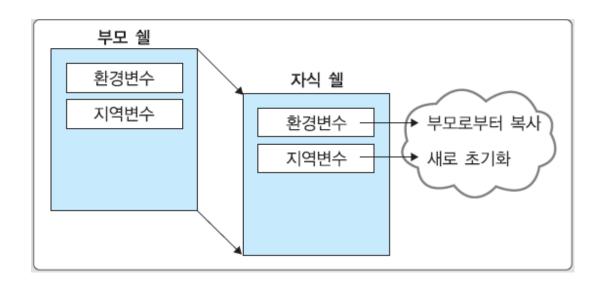
□ 변수를 하나만 사용
$ read x
```

# \$ read x Merry Christmas ! \$ echo \$x Merry Christmas !

# 환경변수와지역변수

#### □ 쉘 변수

- ▶ 환경변수와 지역변수 두 종류로 나눌 수 있음
- ▶ 환경 변수는 값이 자식 프로세스에게 상속되나 지역변수는 그렇지 않음



# 환경변수와지역변수예

```
$ country=대한민국 city=서울
$ export country
$ echo $country $city
대한민국 서울
$ bash # 자식 쉘 시작
$ echo $country $city
대한민국
$ ^D # 자식 쉘 끝
$ echo $country $city
대한민국 서울
```

# 사전정의환경변수

#### □ 그 의미가 미리 정해진 환경변수들

이름	의미
\$USER	사용자 이름
\$TERM	터미널 타입
\$PATH	명령어를 검색할 디렉터리들의 리스트
\$HOME	홈 디렉터리
\$SHELL	로그인 쉘의 경로명
\$MAIL	메일 박스의 경로명
\$HOSTNAME	호스트 이름

```
$ echo 홈 = $HOME 사용자 = $USER 쉘 = $SHELL
홈 = /user/faculty/chang 사용자 = chang 쉘 = /bin/bash
$ echo 터미널 = $TERM 경로 리스트 = $PATH
터미널 = xterm 경로 리스트 = /bin:/usr/bin:/usr/local/bin
```

# 사전 정의 지역 변수

이름	의미
\$\$	쉘의 프로세스 번호
\$0	쉘 스크립트 이름
\$1 ~ \$9	명령줄 인수
\$*	모든 명령줄 인수 리스트
\$#	명령줄 인수의 개수

```
#!/bin/bash
# builtin.bash
echo 이 스크립트 이름: $0
echo 첫 번째 명령줄 인수: $1
echo 모든 명령줄 인수: $*
echo 이 스크립트를 실행하는 프로세스 번호: $$
$ builtin.bash hello shell
이 스크립트 이름: builtin.sh
첫 번째 명령줄 인수: hello
모든 명령줄 인수: hello shell
이 스크립트를 실행하는 프로세스 번호: 1259
```

### Bash스크립트작성및실행과정

### (1) 에디터를 사용하여 Bash 스크립트 파일 작성

```
#!/bin/bash
# state.bash
echo -n 현재 시간:
date
echo 현재 사용자:
who
echo 시스템 현재 상황:
Uptime
```

### (2) chmod를 이용하여 실행 모드로 변경

```
$ chmod +x state.bash
```

### (3) 스크립트 이름을 타입핑하여 실행

```
$ state.bash
```

# if문

```
if 문
if 조건식
then
명령어리스트
fi
```

# □ 조건식[식]

```
□ 여
if [ $# -ne 1 ]
```

```
#!/bin/bash
# 사용법: wc1.bash 파일
# 명령줄 인수 개수를 확인하고 wc 명령어를 실행한다.
if [ $# -ne 1 ]
then
  echo 사용법: $0 파일
  exit 1
fi
file=$1
wc $file
$ wc1.bash
사용법: wc1.bash 파일
$ wc1.bash cs1.txt
38 318 2088 cs1.txt
```

### if-then-else

### □ if-then-else 구문

```
if 조건식
then
명령어리스트
else
명령어리스트
fi
```

```
#!/bin/bash
# 사용법: count1.bash [디렉터리]
# 대상 디렉터리 내의 파일과 서브디렉터리 개수를 프린트
if [ $# -eq 0 ]
then
  dir="."
else
  dir=$1
fi
echo -n $dir 내의 파일과 서브디렉터리 개수:
1s $dir | wc -1
$ count1.bash
. 내의 파일과 서브디렉터리 개수: 17
```

# 비교연산

### □ 비교 연산은 산술 비교 연산, 문자열 비교 연산

산술 비교 연산자	의미
정수1 -eq 정수2	두 정수가 같으면 참 아니면 거짓
정수1 -ne 정수2	두 정수가 다르면 참 아니면 거짓
정수1 -gt 정수2	정수1이 정수2보다 크면 참 아니면 거짓
정수1 -ge 정수2	정수1이 정수2보다 크거나 같으면 참 아니면 거짓
정수1 -lt 정수2	정수1이 정수2보다 작으면 참 아니면 거짓
정수1 -le 정수2	정수1이 정수2보다 작거나 같으면 참 아니면 거짓

# 문자열 비교 연산

문자열	의미
비교 연산자	
문자열1 == 문자열2	두 문자열이 같으면 참 아니면 거짓
문자열1 != 문자열2	두 문자열이 다르면 참 아니면 거짓
-n 문자열	문자열이 null이 아니면 참
-z 문자열	문자열이 null이면 참

```
#!/bin/bash
# 사용법: reply.bash
# 계속 여부를 입력받아 프린트한다.
echo -n "계속 하겠습니까 ?"
read reply
if [ $reply == "예" ]
then
   echo 예
elif [ $reply == "아니오" ]
then
   echo 아니오
fi
$ reply.bash
계속 하겠습니까 ?아니오
아니오
```

# 파일관련연산

파일 관련 연산자	의미
-a 파일 -e 파일	해당 파일이 존재하면 참
-r 파일	사용자가 해당 파일을 읽을 수 있으면 참
-w 파일	사용자가 해당 파일을 쓸 수 있으면 참
-x 파일	사용자가 해당 파일을 실행할 수 있으면 참
-0 파일	사용자가 해당 파일의 소유자이면 참
-z 파일	해당 파일의 크기가 o이면 참
-f 파일	해당 파일이 일반 파일이면 참
-d 파일	해당 파일이 디렉터리이면 참

# 파일 관련 연산: 예

```
if [ -e $file ]
then # $file이 존재하면
wc $file
else # $file이 존재하지 않으면
echo "오류 ! 파일 없음"
fi
```

```
if [ -d $dir ]
then

echo -n $dir 내의 파일과 서브디렉터리 개수:
ls $dir | wc -l
else
echo $dir\: 디렉터리 아님
fi
```

# 부울연산자

#### □ 조건식에 부울 연산자 사용

- ➤! 부정(negation)
- ➤ && 논리곱(logical and)
- ▶ || 논리합(logical or)

```
# $file이 일반 파일이고 쓸수 있으면
if [ -f $file ] && [ -w $file ]
then
  uptime > $file
fi
if [! -e $file]
then # $file이 존재하지 않으면
  echo $file : 파일 없음
fi
if [ ! -d $file ]
then # $dir이 디렉터리가 아니면
  echo $file : 디렉터리 아님
fi
```

# 산술연산

### □ 산술 연산

```
$ a=2+3
$ echo $a
$ a=`expr 2 + 3`
```

### □ let 명령어를 이용한 산술연산

```
$ let 변수=수식
$ let a=2*3
$ echo $a
6
$ let a=$a+2
$ echo $a
8
$ let a*=10
$ let b++
```

# 변수타입선언

### □ 변수 타입 선언: declare

```
$ declare -i a # a는 정수형 변수
$ a=12
$ a=a+1 # let 필요 없음
$ echo $a
$ a=12.3 # 오류 메세지
bash: 12.3: syntax error in
expr(error token is ".3")
$ declare -r b=23.4 # 읽기 전용
$ b=23.5 # 오류 메세지
bash: b: readonly variable
```

이름	의미
declare -r 변수	읽기 전용 변수로 선언
declare -i 변수	정수형 변수로 선언
declare -a 변수	배열 변수로 선언
-ll	스크립트 안에서 정의된
declare -f	모든 함수들을 보여준다.
declare -f 함수이	해당 함수 이름을 보여
름	준다.
declare -x 변수	환경변수로 export

# Bash제어구조

- □조건 if
- □ 스위치 case
- □ 반복 for, while

### 조건문

```
if 조건식
then
  명령어리스트
fi
if 조건식
then
  명령어리스트
else
  명령어리스트
fi
```

```
if 조건식
then
  명령어리스트
elif 조건식
then
  명령어리스트
else
  명령어리스트
fi
```

# 새로운조건식

```
□ 새로운 조건식
 if ((수식))
□ 예
  #!/bin/bash
  # 사용법: wc2.bash
  # 명령줄 인수의 개수를 확인하고 wc 명령어를 실행한다.
   if (( $# != 1 ))
   then
     echo 사용법: $0 파일
     exit 1
   fi
   file=$1
  wc $1
```

# 산술연산자

산술 연산자	의미
-	단일항 음수
!	논리 부정
* / %	곱셈, 나눗셈, 나머지
+ -	덧셈, 뺄셈
<< >>	비트 좌이동, 비트 우이동
<= >= < >	관계 연산
== !=	동등, 비동등
&&	논리합, 논리곱
& ^	비트 and, 비트 xor, 비트 or

# 중첩조건문:예

```
#!/bin/bash
# 사용법: score1.bash
# 점수에 따라 학점을 결정하여 프린트
echo -n '점수 입력: '
read score
if (( $score >= 90 ))
then
  echo A
elif (( $score >= 80 ))
then
  echo B
elif (( $score >= 70 ))
then
  echo C
else
  echo 노력 요함
fi
```

\$score1.bash

점수 입력: 85

В

### 스위치

```
case $변수 in
패턴1) 명령어리스트;;
패턴2) 명령어리스트;;
···
*) 명령어리스트;;
esac
```

```
#!/bin/bash
# 사용법: score2.bash
# 점수에 따라 학점을 결정하여 프
 린트한다.
echo -n '점수 입력: '
read score
let grade=$score/10
case $grade in
  "10" | "9") echo A;;
  "8") echo B;;
  "7") echo C;;
  *) echo 노력 요함;;
esac
```

# 반복문: for

```
□ for 구문

▷ 리스트의 각 값에 대해서 명령어들을 반복

for 이름 in 단어리스트

do

명령어리스트

done
```

```
#!/bin/bash
# 사용법: invite.bash
# 저녁 초대 메일을 보낸다.
invitee=(lee kim choi)
for person in ${invitee[*]}
do
echo "초대의 글 : 오늘 저녁
식사 모임에 초대합니다." | \
mail "${person}@gmail.com"
done
```

### 모든명령줄인수처리

#### □ 모든 명령줄 인수 처리

```
for file in $*
do
...
done
```

```
#!/bin/bash
# 사용법: perm1.bash 파일*
# 파일의 사용권한과 이름을 프린트한다.
if [ $# -eq 0 ]
then
 echo 사용법: $0 파일*
 exit 1
fi
echo " 사용권한 파일"
for file in $*
do
 if [ -f $file ]
  then
   fileinfo=`ls -l $file`
   perm=`echo "$fileinfo"|cut -d' ' -f1`
   echo "$perm $file"
 fi
done
```

# 반<del>복문</del>: while

```
■ while 문

➤ 조건에 따라 명령어들을 반복적으로 실행

while 조건식

do

        명령어리스트

done
```

```
#!/bin/bash
# 사용법: power.bash
# 2의 1승부터 10승까지 프린트
let i=2
let j=1
while (( $j <= 10 ))
do
  echo '2 \wedge' $j = $i
  let i*=2
  let j++
done
```

### menu,bash

```
#!/bin/bash
# 사용법: menu.bash
# 메뉴에 따라 해당 명령어를 실행한다.
echo 명령어 메뉴
cat << MENU
    d : 날짜 시간
    1 : 현재 디렉터리 내용
    w : 사용자 보기
    q : 끝냄
MENU
stop=0
```

```
while (($stop == 0))
do
 echo -n '? '
 read reply
  case $reply in
    "d") date;;
    "1") ls;;
    "w") who;;
    "q") stop=1;;
    *) echo 잘못된 선택;;
 esac
done
```

### menu,bash

```
$ menu.bash
명령어 메뉴
d : 날짜 시간
1 : 현재 디렉터리 내용
w : 사용자 보기
q : 끝냄
? d
2012년 2월 23일 목요일 오후 07시 33분 27초
? q
```

# 함수

```
□ 함수 정의
함수이름()
{
명령어리스트
}
```

□ 함수 호출함수이름 [매개변수]

```
#!/bin/bash
# 사용법: 1shead.bash
1shead() {
 echo "함수 시작, 매개변수 $1"
 date
 echo "디렉터리 $1 내의 처음 3개
 파일만 리스트"
 1s -1 $1 | head -4
echo "안녕하세요"
1shead /tmp
exit 0
```

### 함수

#### \$1shead.bash

```
안녕하세요
함수 시작, 매개변수 /tmp
2012년 2월 23일 목요일 오후 08시 31분 31초
디렉터리 /tmp 내의 처음 3개 파일만 리스트
총 1184
-rw----- 1 chang faculty 11264 2009년 3월 28일 Ex01378
-rw----- 1 chang faculty 12288 2011년 5월 8일 Ex02004
-rw----- 1 root other 8192 2011년 5월 4일 Ex02504
```

### shift

#### □ shift 명령어

- ➤ shift [리스트변수]
- ▶ 명령줄 인수[리스트 변수] 내의 원소들을 하나씩 왼쪽으로 이동

```
#!/bin/bash
# 사용법: perm2.bash 파일*
# 파일의 사용권한과 이름을 프린트
if [ $# -eq 0 ]
then
echo 사용법: $0 files
exit 1
fi
echo " 허가권 파일"
```

```
while [ $# -gt 0 ]
do
   file=$1
   if [ -f $file ]
   then
     fileinfo=`ls -l $file`
     perm=`echo "$fileinfo" |
           cut -d' ' -f1`
     echo "$perm $file"
   fi
   shift
done
```

# 디렉터리 내의 모든 파일 처리

### □ 디렉터리 내의 모든 파일 처리

- ▶ 해당 디렉터리로 이동
- ➤ for 문과 대표 문자 \*를 사용
- ▶ 대표 문자 \*는 현재 디렉터리 내의 모든 파일 이름들로 대치

```
cd $dir
for file in *
do
...
done
```

# 디렉터리 내의 모든 파일 처리: 예

```
#!/bin/bash
# 사용법: count2.bash [디렉터리]
# 대상 디렉터리 내의 파일, 서브디렉터리, 기타 개수를 세서 프린트
if [ $# -eq 0 ]
then
 dir="."
else
                                     echo $dir\:
  dir=$1
                                     cd $dir
                                     for file in *
if [ ! -d $dir ]
                                     do
then
                                        if [ -f $file ]
  echo $0\: $dir 디렉터리 아님
                                        then
  exit 1
                                         let fcount++
                                        elif [ -d $file ]
let fcount=0
                                        then
let dcount=0
                                         let dcount++
let others=0
                                        else
                                         let others++
                                        fi
```

done

echo 파일: \$fcount 디렉터리: \$dcount 기타: \$others

# 리커전(recursion)

- □ 스크립트도 자기 자신 호출 가능
- □ 어떤 디렉터리의 모든 하위 디렉터리에 대해 동일한 작업을 수행할 때 매우 유용함

```
#!/bin/bash
# 사용법 rhead.bash [디렉터리]
# 대상 디렉터리와 모든 하위 디렉터리 내에 있는 파일들의
 헤더를 프린트
cd $1
for file in *
do
  if [ -f $file ]
  then
    echo "======= $file ======="
    head $file
  fi
  if [ -d $file ]
  then
     /home/pi/.../rhead.bash $file
  fi
done
```