

시스템 프로그래밍 (UNIX Shell)

2019.08.19.~2019.08.23

주수홍

강사 소개

- ▶ 이름 : 주수홍
- ▶ 연락처 : 010-5008-2030
- ▶ 메일 : ozroid@gmail.com

- ▶ 개발경력
 - ▶ H-IDS/N-IDS 개발(UNIX/LINUX 기반)
 - ▶ 홈네트워크 시스템
 - ▶ 무선 통신 Firmware
 - ▶ 네트워크 보안 시스템
 - ▶ 펌웨어 보안 시스템
 - ▶ 핸드폰/텔레메틱스
- ▶ 강의이력
 - ▶ C/C++(STL)/Assembly/MFC
 - ▶ JAVA/RFID
 - ▶ 임베디드(ARM/ATMega) Firmware

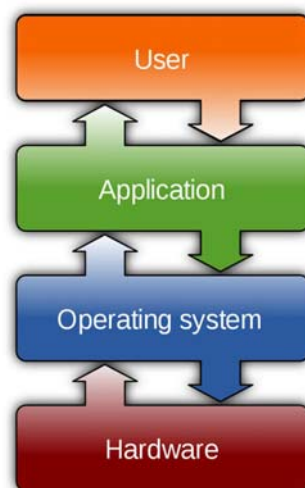
목차

- ▶ 운영체제 개요 및 구조
- ▶ Virtual Box 설치
- ▶ 파일 시스템 개요
- ▶ 파일 편집
- ▶ 실습 : 경로 이동
- ▶ 실습 : 파일 관리
- ▶ 실습 : 쉘 스크립트 기초

운영체제 개요 및 구조

운영체제(OS : Operating System)

- ▶ 운영체제란 무엇인가?
 - ▶ 컴퓨터 시스템의 자원을 관리하는 프로그램의 집합
 - ▶ 사용자와 응용 프로그램을 위하여 자원을 관리 분배
- ▶ 사용의 편리성을 위해 개발됨
 - ▶ 과학자의 소유물에서 일반 사용자로의 이행
 - ▶ 복잡 다변화 되는 컴퓨터의 원활한 이용을 위해
 - ▶ HW의 발전과 더불어 많은 기능 추가
 - ▶ 소형 기기의 성능 향상에 따른 OS 도입 증가



운영체제 개요 및 구조

운영체제(OS : Operating System)

▶ 운영체제의 역할

- ▶ 사용자와의 인터페이스를 정의
- ▶ 사용자들 간에 하드웨어를 공동 사용토록 하고
- ▶ 사용자들 간에 데이터를 공유토록 하며
- ▶ 사용자들 간의 자원 스케줄링
- ▶ 입출력 보조 역할
- ▶ 에러(error) 처리

▶ 운영체제가 관리하는 주된 자원(resource)

- ▶ 프로세서, 기억장치, 입출력 장치, 데이터

운영체제 개요 및 구조

운영체제(OS : Operating System)

▶ 운영체제의 시작

- ▶ 1952년 : GE연구소에서 IBM 701용 SW 개발
- ▶ 1955년 : GE연구소와 노스아메리칸 항공사에서 IBM 704용 SW 개발
- ▶ 1966년 : OS/360
 - ▶ IBM : 1964년 System/360 개발 후 OS/360 개발
 - ▶ 시스템/360 계열 컴퓨터를 이용하기 위해 개발한 운영체제
 - ▶ 운영체제 개념 확립
- ▶ 1976년 : UNIX 등장
 - ▶ 현재 사용되는 OS에 막대한 영향을 미침
 - ▶ 가장 안정적이고 강력한 성능을 보유했다고 평가됨
 - ▶ 국가 기간망, 금융 시스템에 사용

운영체제 개요 및 구조

유닉스(UNIX)

▶ 시초와 발전사

▶ 1969년

- ▶ Bell 연구소의 Ken Thompson이 PDP-7 위에서 개발
- ▶ Dennis Ritchie가 참가하여 대부분을 C언어로 작성



Ken



Dennis

운영체제 개요 및 구조

유닉스(UNIX)

▶ 시초와 발전사

▶ 1973년

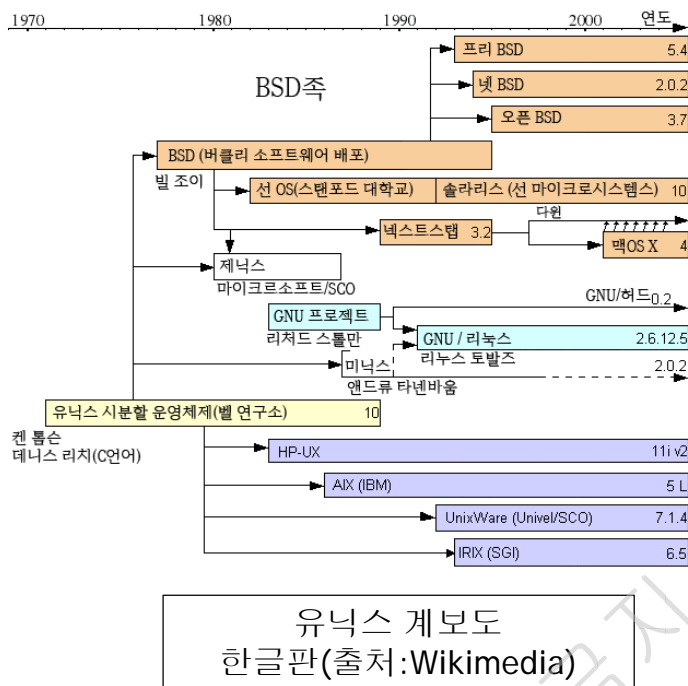
- ▶ SOSP(Symposium of Operating Systems Principles) 컨퍼런스에 소개
- ▶ 버클리 밥 패브리(Bob Fabry) 교수 : 켄 톰프슨과 데니스 리치에게 UNIX 복사본 요청

▶ 1976년

- ▶ Sixth Edition 개발
- ▶ 다른 기관이 사용할 수 있도록 배포한 최초의 버전
- ▶ 켄 톰프슨이 학생을 상대로 안식년 동안 UNIX 교육 실시
- ▶ 이후에 UNIX는 크게 분류해서 연구용, AT&T 계열, BSD(Berkeley Software Distribution) 계열로 분화해서 발전

운영체제 개요 및 구조

유닉스(UNIX)



운영체제 개요 및 구조

유닉스(UNIX)

▶ 시초와 발전사

- ▶ 1979년
 - ▶ Seventh Edition : Bourne shell 개발
- ▶ 1980년
 - ▶ 미국방성 고등 방위 연구 계획국(DARPA) : BSD 향상에 대한 계약 체결(버클리대 밥 패브리 교수)
- ▶ BSD : CSRG(Computer Systems Research Group) 연구 그룹 결성
 - ▶ 군수 업체 BBN Technologies가 연구 중이던 TCP/IP 코드를 개선하여 BSD 반영 시작
- ▶ 1984년
 - ▶ AT&T와 CSRG 갈등 시작 : AT&T의 UNIX 라이선스 문제
- ▶ 1989년
 - ▶ BSD 라이선스 배포 : AT&T 라이선스 회피 및 GNU에 영향
- ▶ 1991년
 - ▶ AT&T 코드가 거의 제거된 무료 BSD 배포 시작 : AT&T와 분쟁 돌입
 - ▶ AT&T와 CSRG 서로 맞고소
- ▶ 1994년
 - ▶ 분쟁 조정 완료 : AT&T 잔여 코드 제거 조건

운영체제 개요 및 구조

유닉스(UNIX)

▶ 계열별 발전사

▶ AT&T UNIX(aka System V)

- ▶ System V, Release 3 개발 (1987년) : STREAMS 도입
- ▶ System V, Release 4 개발 (1989년) : C/Korn shell, Job control, Symbolic link



운영체제 개요 및 구조

유닉스(UNIX)

▶ 계열별 발전사

▶ BSD UNIX(aka Berkeley Unix)

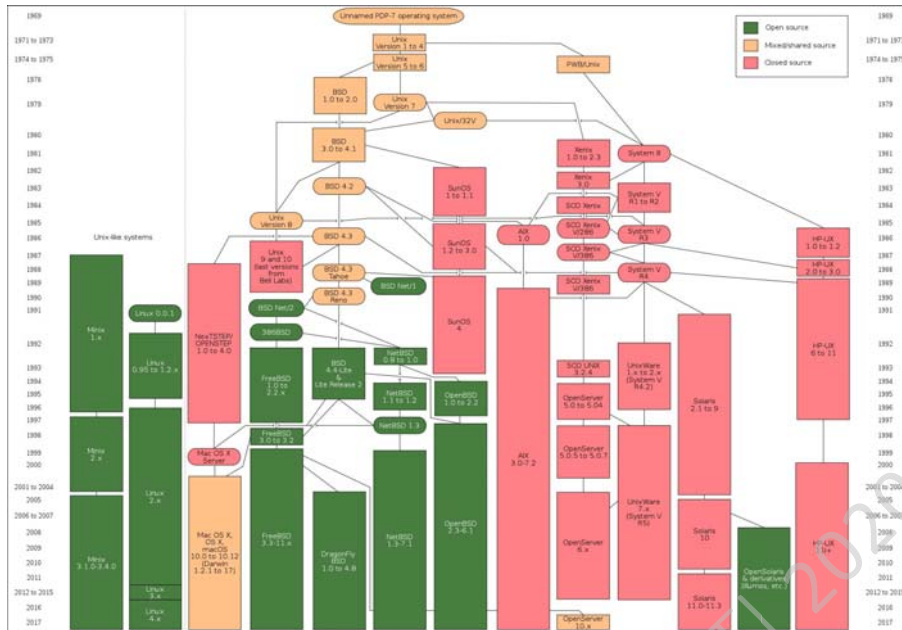
- ▶ BSD 4 개발 (1980년) : Job control, Reliable signal
- ▶ BSD 4.4 개발 (1993년)



FreeBSD



유닉스(UNIX)



유닉스 계보도
(출처:wikipedia)

유닉스(UNIX)

▶ 현재의 대중적 OS

► MacOS

| 버전 | 출시 날짜 |
|---------------|---------------|
| 랩소디 디벨로퍼 릴리즈 | 1997년 08월 31일 |
| 맥 OS X 서버 1.0 | 1999년 03월 16일 |
| 맥 OS X v10.0 | 2001년 03월 24일 |
| 맥 OS X 팬서 | 2003년 10월 24일 |
| 맥 OS X 타이거 | 2005년 04월 29일 |
| 맥 OS X 레퍼드 | 2007년 10월 26일 |
| 맥 OS X 스노 레퍼드 | 2009년 08월 28일 |
| 맥 OS X 라이언 | 2011년 07월 20일 |
| OS X 마운틴 라이언 | 2012년 07월 25일 |
| OS X 매버릭스 | 2013년 10월 22일 |
| OS X 요세미티 | 2014년 10월 16일 |
| OS X 엘카피텐 | 2015년 10월 01일 |
| macOS 시에라 | 2016년 09월 20일 |
| macOS 하이 시에라 | 2017년 09월 26일 |
| macOS 모하비 | 2018년 09월 24일 |

운영체제 개요 및 구조

유닉스(UNIX)

▶ 현재의 대중적 OS

▶ Windows

- ▶ 텍스트 기반의 Dos에서 GUI 체제로 이행하면서 개발된 OS
- ▶ 초기에는 데스크톱용으로 개발되었지만 병행 개발되던 서버 기술인 NT를 적용하여 현재에는 NT서버와 통합됨



운영체제 개요 및 구조

유닉스(UNIX)

▶ 현재의 대중적 OS

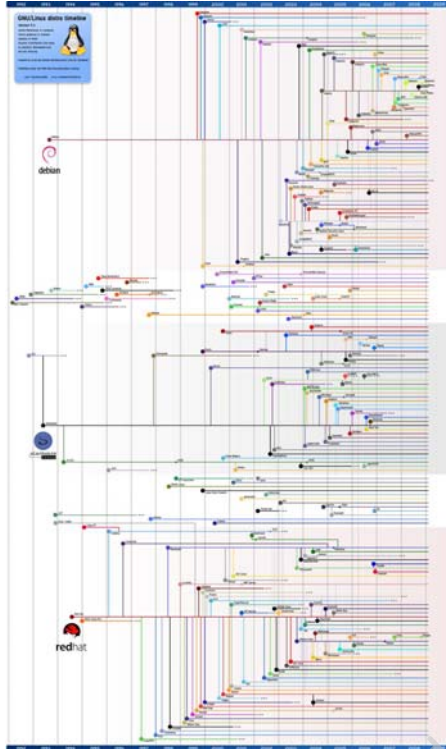
▶ Linux

- ▶ 리누스 토발즈(Linus Benedict Torvalds)와 GNU 프로젝트에 의해 탄생된 신종 OS
- ▶ 무료로 배포되고 있으며 네트워크 기기와 핸드폰 등 IT분야에 전방위 적으로 확산 됨



운영체제 개요 및 구조

유닉스(UNIX)



리눅스 계보도

운영체제 개요 및 구조

유닉스(UNIX)

▶ 현재의 대중적 OS

- ▶ DOS(Disk Operation System) : CP/M(Control Program/Monitor)
 - ▶ 1973년 디지털 리서치 : 게리 킬달
 - ▶ 70년대 PC(Personal Computer : 개인용 컴퓨터)의 운영체제 시장 선점
 - ▶ 존재하던 운영체제 중 가장 많이 판매(100만장)
 - ▶ 거의 모든 인텔 8080 및 자일로그 Z80 기반의 컴퓨터에서 동작 가능하였음



운영체제 개요 및 구조

유닉스(UNIX)

▶ 현재의 대중적 OS

▶ DOS(Disk Operation System) : MS-DOS

- ▶ IBM : 게리 킬달과의 협상 결렬 후 MS사의 빌 게이츠와 접촉
- ▶ 빌 게이츠는 시애틀 컴퓨터사(Seattle Computer Products)의 팀 패터슨으로부터 Q-DOS를 사들여서 IBM에 납품



운영체제 개요 및 구조

유닉스(UNIX)

▶ 현재의 대중적 OS

▶ DOS(Disk Operation System) : MS-DOS 연대기

| 연도 | 버전 | 비고 |
|-------|------|--------------|
| 1981년 | 1.0 | PC-DOS로 명칭 |
| 1982년 | 1.25 | MS-DOS 명칭 사용 |
| 1983년 | 2.0 | /를 \로 대체 |
| 1984년 | 3.0 | AT 지원 |
| 1988년 | 4.0 | |
| 1991년 | 5.0 | 윈도우즈 3.0 |
| 1993년 | 6.0 | 윈도우즈 3.1 |
| 1995년 | 7.0 | 윈도우즈 95 |
| 1998년 | 7.1 | |
| 2000년 | 8.0 | 윈도우즈 ME |

Virtual Box 설치

개요

▶ 가상머신

- ▶ 실제로 존재하지 않는 컴퓨터를 만드는 개념
- ▶ CPU/RAM/HDD 등 컴퓨터의 주요 부품을 가상으로 생성
- ▶ 가상의 공간 안에서 프로그램을 구동
- ▶ 근래에는 HW적으로 가상머신을 지원하기 시작
 - ▶ Intel : VT-x
 - ▶ AMD : AMD-V, SVM

▶ 기타

- ▶ 에뮬레이션 : 모든 부품의 기능을 SW만으로 구현하는 방식
- ▶ HW 가상화 : 주요 부품을 HW의 기능을 통해 지원 받아 구현하는 방식
- ▶ SW 가상화/SW Container : 주요 부품을 OS의 지원 하에서 구현하는 방식
- ▶ 반가상화 : 설치된 OS의 수정이나 전용 드라이버를 통해 주요 부품을 구현하는 방식

Virtual Box 설치

개요

▶ Virtual Box 소개

- ▶ 개발사 : 이노텍(InnoTek)
- ▶ 썬 마이크로시스템즈에 인수되어 현재는 오라클 소유
- ▶ 바이너리는 GPLv2로 배포 중
- ▶ 확장팩의 경우 개인 한정 평가 라이선스로 배포 중(상업용 사용 불가)
- ▶ 게스트 확장 기능 : 성능 향상 등의 기능을 추가적으로 설치 가능
- ▶ HW 가상화 기능을 사용하여 구현됨

Virtual Box 설치

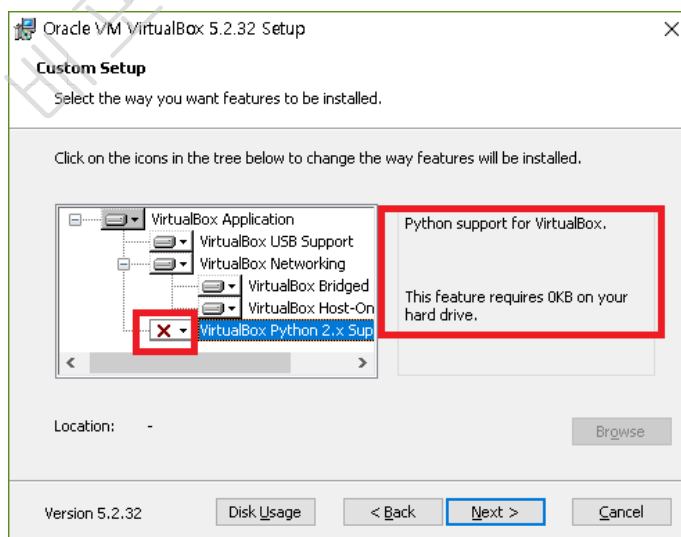
설치시작

▶ 설치 시작 : VirtualBox-5.2.32-132073-Win.exe



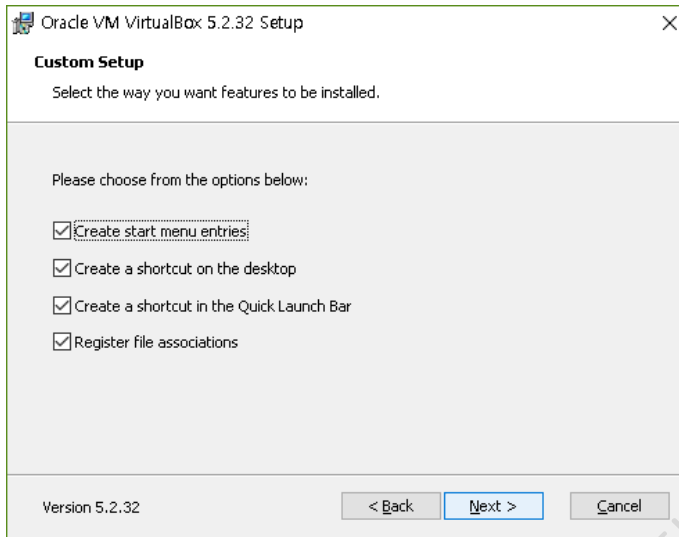
Virtual Box 설치

설치시작



Virtual Box 설치

설치시작



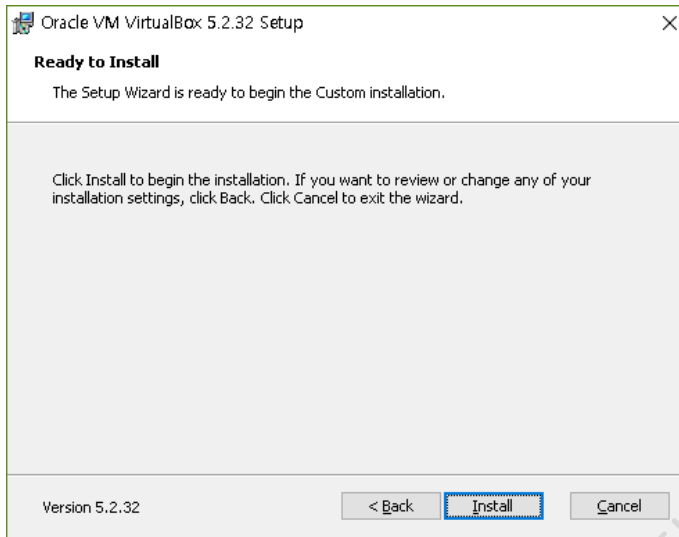
Virtual Box 설치

설치시작



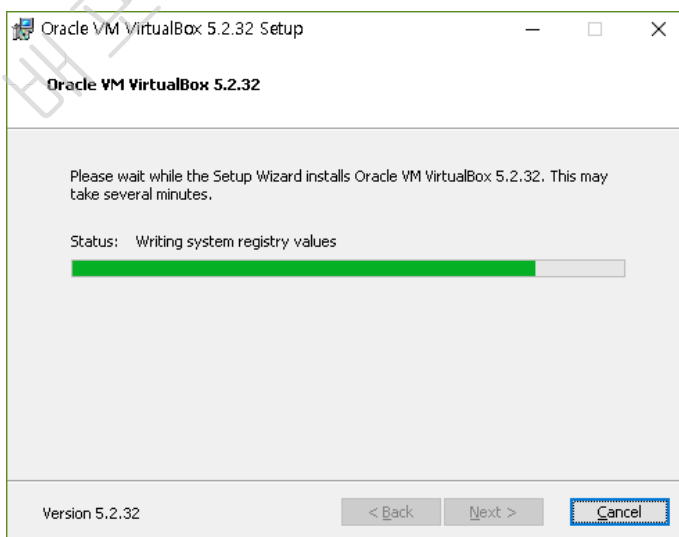
Virtual Box 설치

설치시작



Virtual Box 설치

설치시작



Virtual Box 설치

설치시작



Virtual Box 설치

설치시작



Virtual Box 설치

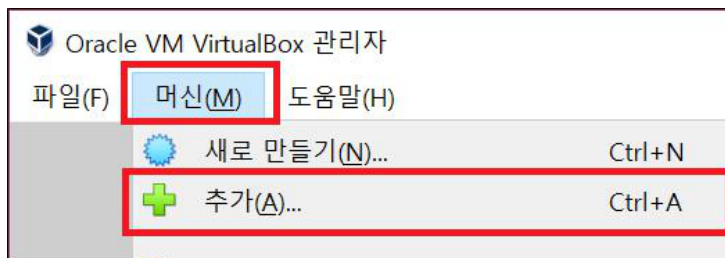
설치 완료



Virtual Box 설치

가상머신 추가

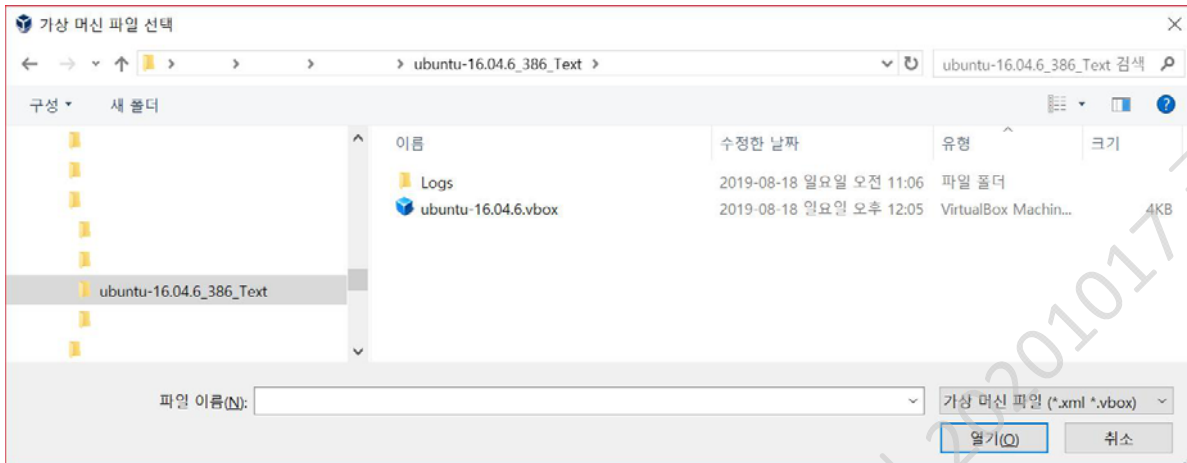
▶ 가상머신 추가



Virtual Box 설치

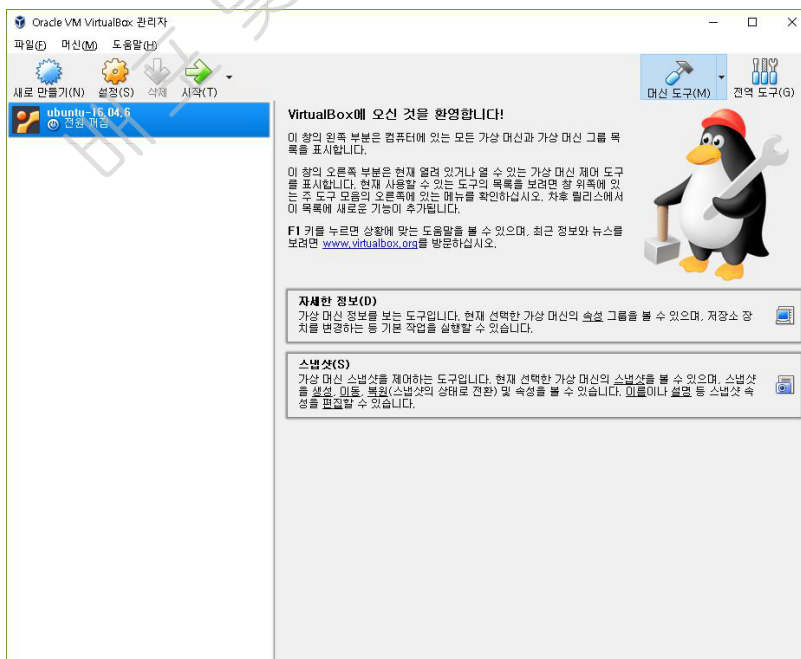
가상머신 파일 선택

▶ 가상머신 파일 선택



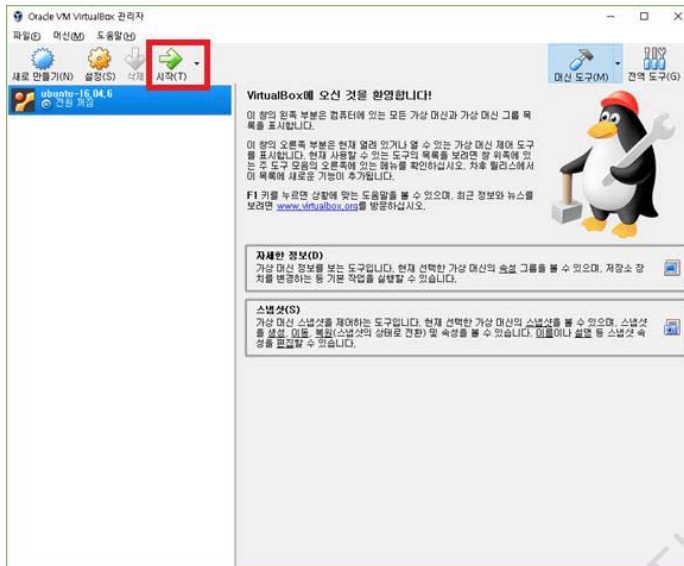
Virtual Box 설치

가상머신 추가 완료



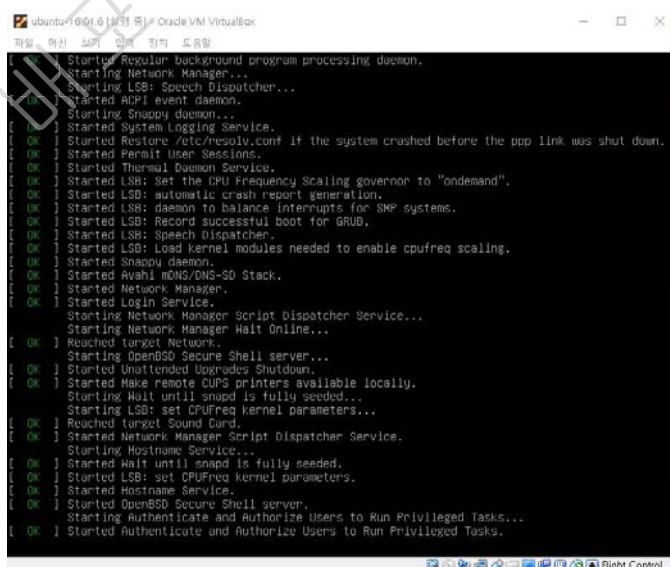
Virtual Box 설치

부팅 시작



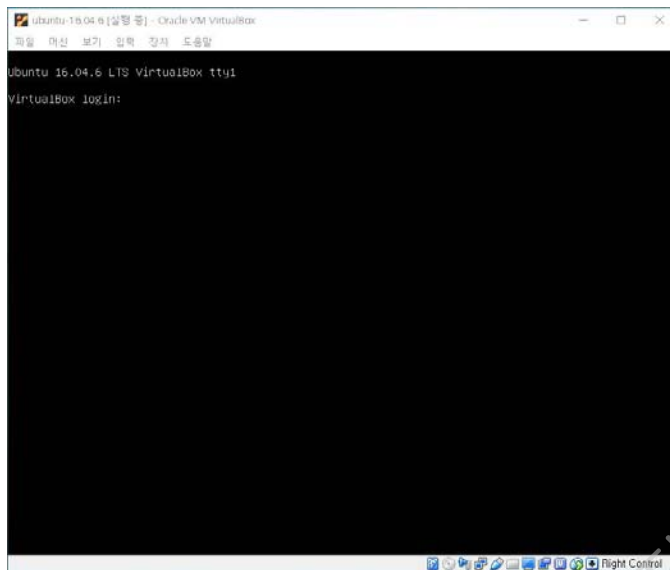
Virtual Box 설치

부팅 시작



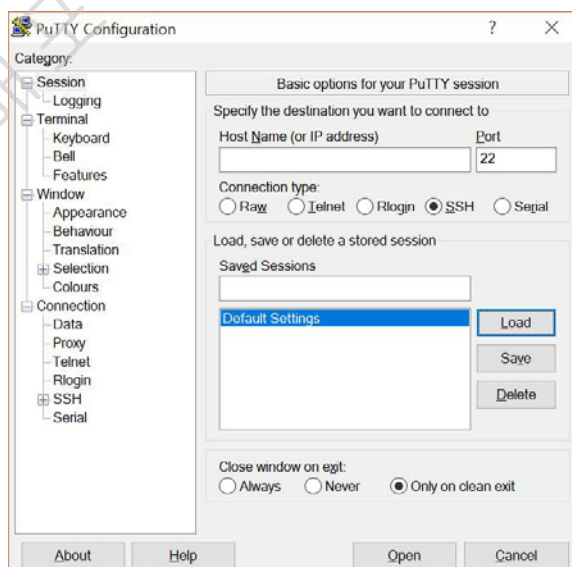
Virtual Box 설치

부팅 완료



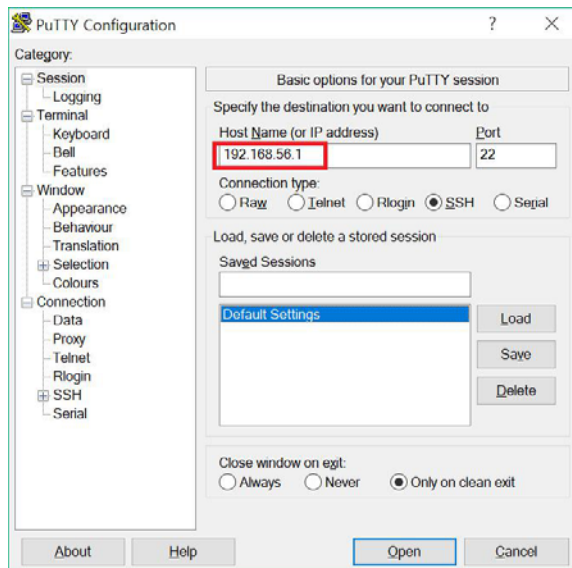
Virtual Box 설치

원격 접속 : putty32.exe 실행



Virtual Box 설치

원격 접속 : IP 입력 후 open 클릭하여 원격 접속



Virtual Box 설치

원격 접속 : 로그인

로그인

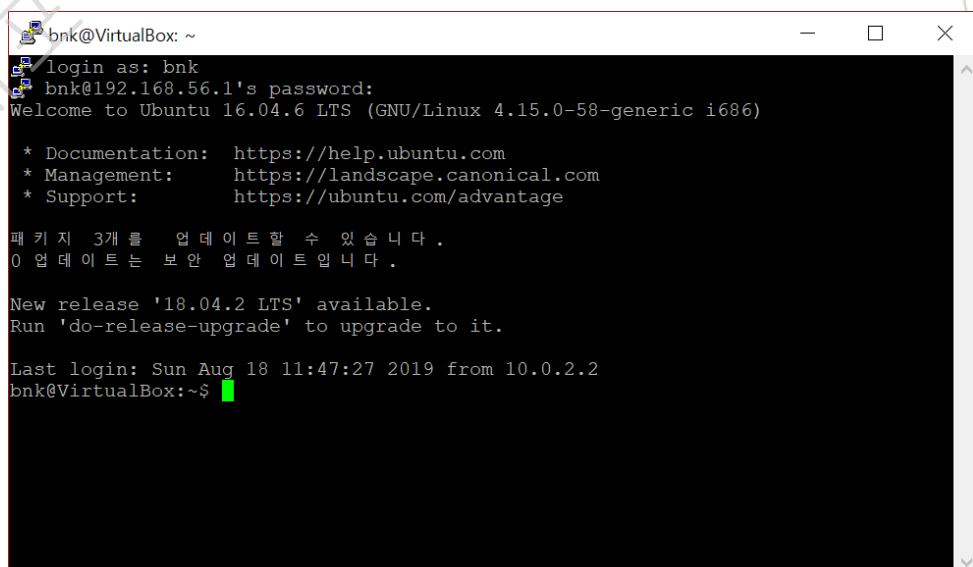
- ▶ ID : bnk
- ▶ pw : 1
- ▶ 대소문자 유의
- ▶ 패스워드 입력 시 글자가 보이지 않음

로그아웃 방법 1

- ▶ Ctrl + D

로그아웃 방법 2

- ▶ logout

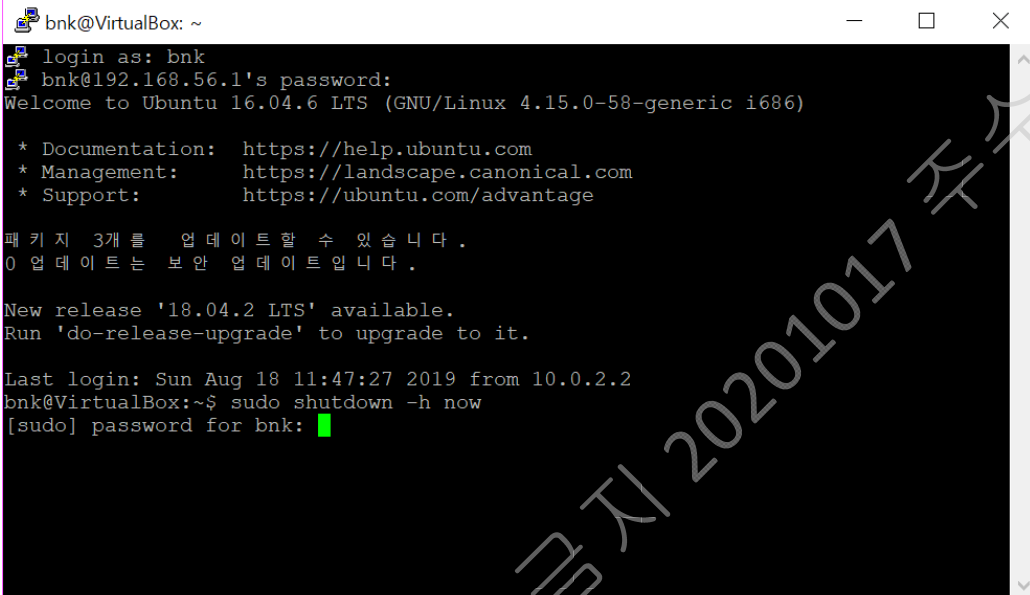


Virtual Box 설치

원격 접속 : 시스템 종료 명령의 사용

▶ `sudo shutdown -h now`

▶ 패스워드 1 입력



```
bnk@VirtualBox: ~  
login as: bnk  
bnk@192.168.56.1's password:  
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-58-generic i686)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
패키지 3개를 업데이트할 수 있습니다.  
0 업데이트는 보안 업데이트입니다.  
  
New release '18.04.2 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Sun Aug 18 11:47:27 2019 from 10.0.2.2  
bnk@VirtualBox:~$ sudo shutdown -h now  
[sudo] password for bnk: █
```

파일 시스템 개요

파일 기반 제어 방식

▶ 모든 자원을 파일로 취급

▶ 파일과 Device

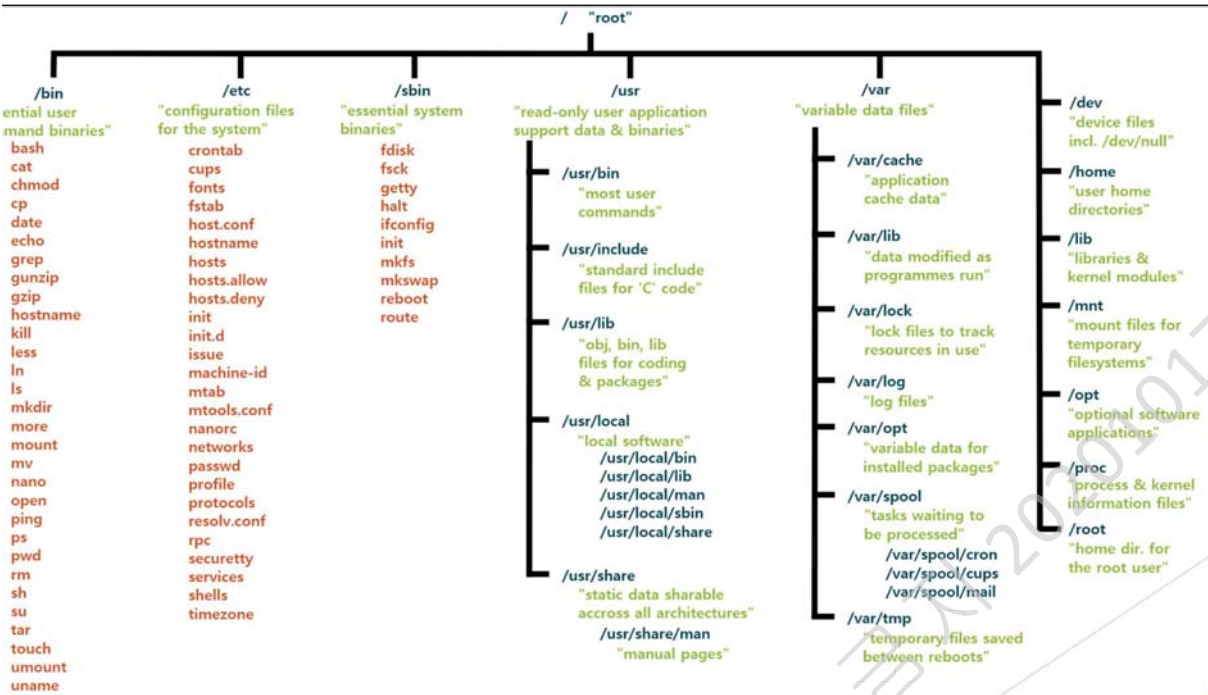
- ▶ 일반 파일과 동일하게 장치 및 구성 요소들을 모두 파일로 취급
- ▶ 파일로의 추상화를 통해 프로그래머 및 관리자들에게 일관성 있는 접근 방식 제공

▶ 디렉터리

- ▶ 기능과 용도에 따라 Directory Tree 구성

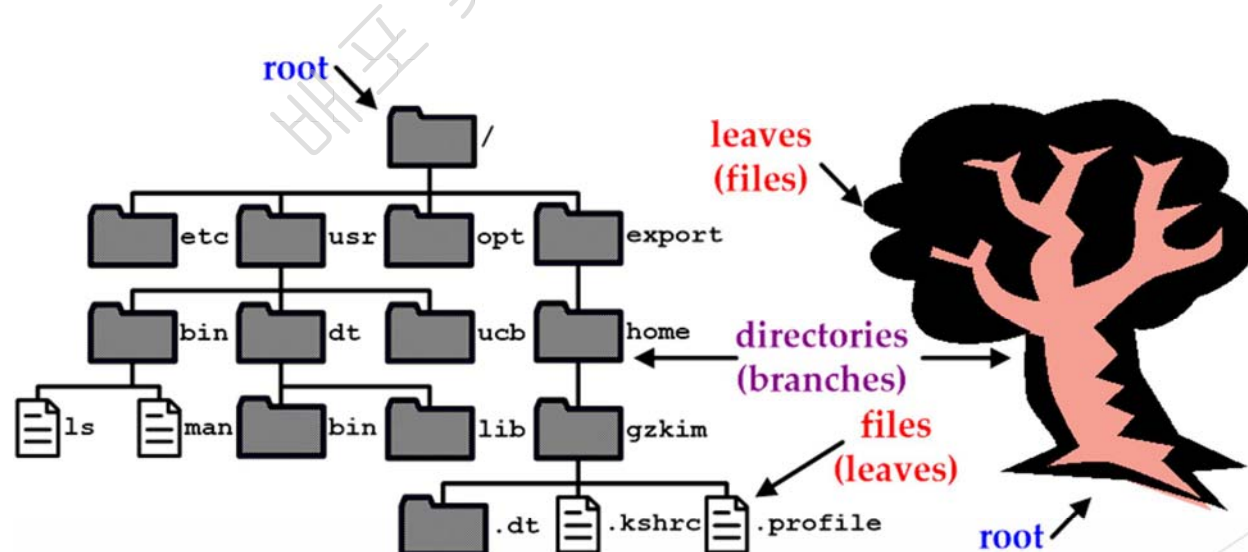
파일 시스템 개요

파일 기반 제어 방식 : UNIX Directory Tree(출처:Wikimedia)



파일 시스템 개요

파일 기반 제어 방식 : Directory Tree의 유래



파일 시스템 개요

주요 디렉터리1(Solaris 기준)

- ▶ **root(/)** : root directory
 - ▶ 최상위 디렉터리의 호칭
 - ▶ 파일 시스템 구조의 시작점을 의미
- ▶ **/usr**
 - ▶ 실행 가능한 명령어, 시스템 관리자용 유틸리티 명령어, 라이브러리 등이 위치
- ▶ **/opt**
 - ▶ 추가적으로 설치되는 응용 프로그램 등이 위치
- ▶ **/etc**
 - ▶ 시스템 관리용 파일 등이 위치
- ▶ **/export/home**
 - ▶ 일반 사용자의 홈 디렉터리가 위치

파일 시스템 개요

주요 디렉터리2(Ubuntu 18.04.2 Linux 기준)

```
root@Linux: /
root@Linux: /# ls -al
합계 1436596
drwxr-xr-x 25 root root 4096 7월 30 13:58 .
drwxr-xr-x 25 root root 4096 7월 30 13:58 ..
drwxr-xr-x 2 root root 4096 7월 30 18:00 bin
drwxr-xr-x 4 root root 4096 7월 30 15:07 boot
drwxr-xr-x 2 root root 4096 7월 30 13:47 cdrom
drwxr-xr-x 18 root root 4100 8월 12 16:59 dev
drwxr-xr-x 127 root root 12288 7월 30 18:23 etc
drwxr-xr-x 3 root root 4096 7월 30 13:48 home
lrwxrwxrwx 1 root root 33 7월 30 13:58 initrd.img -> boot/initrd.img-4.18.0-15-generic
lrwxrwxrwx 1 root root 33 7월 30 13:57 initrd.img.old -> boot/initrd.img-4.18.0-15-generic
drwxr-xr-x 21 root root 4096 7월 30 14:04 lib
drwxr-xr-x 2 root root 4096 2월 10 2019 lib64
drwx----- 2 root root 16384 7월 30 13:34 lost+found
drwxr-xr-x 2 root root 4096 2월 10 2019 media
drwxr-xr-x 2 root root 4096 2월 10 2019 mnt
drwxr-xr-x 2 root root 4096 2월 10 2019 opt
dr-xr-xr-x 213 root root 0 8월 12 16:58 proc
drwx----- 5 root root 4096 7월 30 17:09 root
drwxr-xr-x 28 root root 920 8월 12 17:00 run
drwxr-xr-x 2 root root 12288 7월 30 15:00 sbin
drwxr-xr-x 12 root root 4096 7월 30 16:26 snap
drwxr-xr-x 2 root root 4096 2월 10 2019 srv
-rw----- 1 root root 1470955520 7월 30 13:34 swapfile
dr-xr-xr-x 13 root root 0 8월 12 17:01 sys
drwxrwxrwt 15 root root 4096 8월 12 17:01 tmp
drwxrwxr-x 2 root root 4096 7월 30 13:40 ubiquity-apt-clone
drwxr-xr-x 11 root root 4096 7월 30 15:00 usr
drwxr-xr-x 15 root root 4096 7월 30 17:52 var
lrwxrwxrwx 1 root root 30 7월 30 13:58 vmlinuz -> boot/vmlinuz-4.18.0-15-generic
root@Linux: /#
```

파일 시스템 개요

주요 디렉터리2(Ubuntu 18.04.2 Linux 기준)

| 디렉터리 | 설명 |
|------------|---|
| bin | 기본적인 사용을 위한 명령어 파일이 위치 |
| boot | 부트로더 등 부팅에 관련된 파일이 위치 |
| dev | device 접근에 관련된 파일이 위치 |
| etc | 시스템 관리용 파일 등이 위치 |
| home | 일반 사용자들의 홈 디렉터리가 생성되는 위치 |
| lib | 커널 모듈 및 각종 주요 라이브러리 파일들이 위치 |
| lost+found | 파일 시스템에 문제가 발생할 경우 관련된 파일이 생성되는 위치 |
| media | 외부 장치 등을 연결할 때 사용하는 디렉터리 |
| mnt | 장치나 리소스 등을 파일로 연결할 때 사용하는 디렉터리 |
| opt | 추가적으로 설치되는 응용 프로그램 등이 위치 |
| proc | 실행중인 프로그램에 대한 정보가 들어 있는 디렉터리 |
| root | 관리자 계정의 root 계정의 홈 디렉터리 |
| run | 실행중인 서비스와 관련된 파일이 위치 |
| sbin | 관리자 전용 실행 파일 등 시스템에 관련된 파일이 위치 |
| snap | 샌드박스 패키지용 디렉터리 |
| srv | 서버 프로그램들이 외부에 서비스를 할 경우 이용되는 디렉터리 |
| sys | 리눅스 커널 시스템 파일들이 위치 |
| tmp | 임시 파일들이 저장되는 디렉터리 |
| usr | Unix System Resource 일반적인 실행파일/라이브러리/헤더 파일 등이 위치 |
| var | 자주 사용되면서 그 값이 변경되는 파일 등이 위치(로그 등) |

파일 시스템 개요

Path : 경로

▶ 개념

- ▶ 유닉스 파일 시스템에서 모든 파일은 Tree 구조로 존재함
- ▶ 이러한 파일이나 폴더들의 고유 위치를 의미
- ▶ 구분자 : Delimiter
 - ▶ 디렉터리와 디렉터리 사이를 표시할 때 구분 짓기 위한 기호
 - ▶ 유닉스/리눅스/MacOS : /
 - ▶ 윈도우즈/Dos(1983) : \



파일 시스템 개요

Path : 경로

▶ 상위/하위 디렉터리

- ▶ A라고 하는 디렉터리를 가지고 있는 B 디렉터리를 상위 디렉터리라고 함
- ▶ 이때의 A 디렉터리를 하위 디렉터리라고 함

▶ . 과 .. 디렉터리

- ▶ . 디렉터리는 현재 작업 중인 디렉터를 나타냄
- ▶ .. 디렉터리는 현재 작업 중인 디렉터리의 상위 디렉터를 나타냄

```
root@Linux: ~  
root@Linux:~# ls -al  
drwx----- 5 root root 4096 7월 30 17:09 .  
drwxr-xr-x 25 root root 4096 8월 12 17:10 ..  
-rw----- 1 root root 1177 8월 12 17:42 .bash_history  
-rw-r--r-- 1 root root 3123 7월 30 15:25 .bashrc  
drwx----- 2 root root 4096 7월 30 14:28 .cache  
drwx----- 3 root root 4096 7월 30 14:28 .gnupg  
-rw-r--r-- 1 root root 148 8월 18 2015 .profile  
drwx----- 2 root root 4096 7월 30 15:58 .ssh  
-rw----- 1 root root 5545 7월 30 17:09 .viminfo  
root@Linux:~#
```

파일 시스템 개요

Path : 경로

▶ 절대 경로와 상대 경로

- ▶ 절대 경로 : 경로 표현 시 /(루트 디렉터리)를 제일 앞에 사용한 경로
- ▶ 상대 경로 : /(루트 디렉터리)를 제일 앞에 사용하지 않고 표시한 경로
- ▶ 사용자의 편의성에 따라 선택적으로 사용됨

파일 시스템 개요

Path : 경로

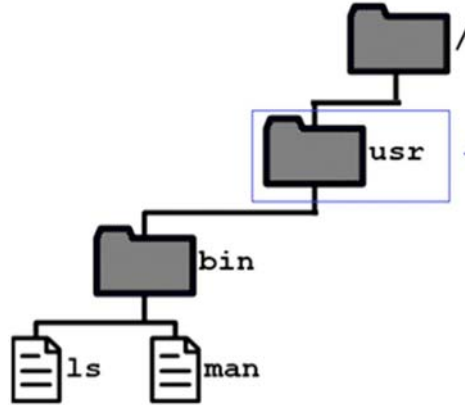
▶ 절대 경로와 상대 경로

▶ man 파일의 경로 예시

- ▶ 상대경로1 : ../bin/man
- ▶ 상대경로2 : bin/man
- ▶ 절대경로 : /usr/bin/man

▶ ls 파일의 경로 예시

- ▶ 상대경로1 : ../bin/ls
- ▶ 상대경로2 : bin/ls
- ▶ 절대경로 : /usr/bin/ls



현재
디렉터리

파일 시스템 개요

Path : 경로

▶ 파일 이름

- ▶ 사람의 이름처럼 이름+분류 형태의 파일 이름을 사용
- ▶ 예 : test.sh
- ▶ 파일 이름 : test
- ▶ 파일 이름과 분류의 분리 기호 : .(dot)을 사용
- ▶ 파일 종류 : sh파일(Shell 파일)

파일 시스템 개요

Path : 경로

▶ 디렉터리 관련 기본 명령어 실습

▶ ls : File List, 특정 디렉터리의 내용을 볼 때 사용

▶ 용법 : ls [-옵션] [파일/디렉터리]

▶ 옵션

- ▶ a : 숨겨진 파일(이름이 .으로 시작하는 파일)을 포함한 모든 파일 출력
- ▶ A : .과 .. 디렉터리를 제외한 숨겨진 파일을 포함한 모든 파일 출력
- ▶ d : 디렉터리 정보 출력
- ▶ l : 상세 정보 출력
- ▶ R : Recursive의 약자로 현재 디렉터리에서 하위 디렉터리가 존재하는 경우 모든 하위 디렉터리의 정보를 출력

▶ 옵션의 경우 여러 개를 겹쳐서 지정 가능

파일 시스템 개요

Path : 경로

▶ A : 파일의 종류를 나타냄

- ▶ - : 일반 파일
- ▶ d : 디렉터리
- ▶ c : 장치 파일(character device)
- ▶ b : 장치 파일(block device)
- ▶ l : 링크 파일
- ▶ s : socket 파일
- ▶ p : pipe 파일

▶ B : 접근 권한을 나타내며 모두 9자리로 구성됨

- ▶ r : read 가능
- ▶ w : write 가능
- ▶ x : execute 가능
- ▶ - : 알파벳 자리에 이 기호가 채워져 있는 경우 해당 기능이 불가능함을 나타냄
- ▶ 3자리 단위로 나누어 짐
 - ▶ 첫 번째 3자리 : 파일의 소유자에 대한 접근 권한
 - ▶ 두 번째 3자리 : 파일의 소유 그룹에 대한 접근 권한
 - ▶ 세 번째 3자리 : 파일의 소유자 및 소유 그룹이 아닌 사용자에게 대한 접근 권한

```
drwxrwxr-x  2 apple apple 4096  7월 30 17:11 code/
```

A B C D E F G H

ls -al의 출력 예시

- ▶ C : 하드 링크 개수
- ▶ D : 파일의 소유자 ID
- ▶ E : 파일의 소유 그룹 ID
- ▶ F : 크기(바이트)
- ▶ G : 가장 최근에 수정된 시간
- ▶ H : 이름

파일 시스템 개요

Path : 경로

▶ 디렉터리 관련 기본 명령어 실습

▶ 실습

- ▶ 현재 디렉터리에서 위 명령을 각각 실행
- ▶ 현재 디렉터리에서 위 명령을 중복해서 실행

파일 시스템 개요

Path : 경로

▶ 디렉터리 관련 기본 명령어 실습

▶ pwd : 현재 작업 디렉터리를 확인할 때 사용, 현재 디렉터리의 절대 경로 출력

- ▶ 용법 : pwd
- ▶ 실습
 - ▶ 명령을 실행

▶ cd : Change Directory, 작업 디렉터를 이동할 때 사용

- ▶ 용법 : cd [상대/절대 디렉터리명]
- ▶ 실습
 - 1) c로 시작하는 디렉터를 확인
 - 2) c로 시작하는 디렉터리로 상대 경로를 사용하여 이동
 - 3) 2)의 상태에서 현재 디렉터리 확인
 - 4) 3)의 상태에서 상대 경로를 사용하여 상위 디렉터리로 이동
 - 5) 1)의 상태에서 c로 시작하는 디렉터리로 절대 경로를 사용하여 이동
 - 6) 5)의 상태에서 절대 경로를 사용하여 상위 디렉터리로 이동
 - 7) 2)의 상태에서 cd만 입력한 후 엔터 입력 후 pwd로 현재 디렉터리 확인

파일 시스템 개요

Path : 경로

▶ 디렉터리 관련 기본 명령어 실습

▶ mkdir : Make Directory, 디렉터리를 생성할 때 사용

- ▶ 용법 : mkdir [-옵션] 디렉터리명1 [디렉터리명2] [디렉터리명3] [디렉터리명4]...
- ▶ 옵션
 - ▶ p : 생성할 디렉터리의 경로에서 상위 디렉터리가 존재하지 않으면 필요한 중간 디렉터리를 자동으로 생성
 - ▶ 추가 디렉터리명 : 동시에 여러 디렉터리 생성 가능
- ▶ 실습
 - ▶ test 디렉터리를 생성하고 확인
 - ▶ 현재 디렉터리에서 ./aaaa/bbbb에 cccc 디렉터를 생성하고 확인
 - ▶ 현재 디렉터리에서 a1, a2, a3 디렉터를 한꺼번에 생성하고 확인

파일 시스템 개요

Path : 경로

▶ 디렉터리 관련 기본 명령어 실습

▶ rmdir : Remove Directory, 디렉터리를 삭제할 때 사용(디렉토리 비어 있어야 함)

- ▶ 용법 : rmdir [-옵션] 디렉터리명
- ▶ 옵션
 - ▶ p : 삭제할 디렉터리의 부모 디렉터리가 비어 있으면 부모 디렉터리도 삭제
- ▶ 실습
 - ▶ test 디렉터를 삭제하고 확인
 - ▶ 현재 디렉터리에서 aaaa/bbbb에 cccc 디렉터를 삭제하고 확인
 - ▶ 현재 디렉터리에서 a1, a2, a3 디렉터를 한번에 삭제하고 확인

파일 편집

vi 편집기

- ▶ Unix 초창기에 개발된 멀티 라인 편집기
- ▶ vi : Visual editor의 약어
- ▶ 개발자 : 빌 조이(Bill Joy)
 - ▶ 라인 단위로 편집기를 사용하던 중 불편함을 느껴 개발
 - ▶ 버클리대 소속으로 BSD 개발에 참여
 - ▶ Sun Microsystems의 공동 창립자



파일 편집

vi 편집기

- ▶ vim : Vi IMproved
 - ▶ 개발자 : 브람 무레나르(Bram Moolenaar)
 - ▶ vi 대비 향상된 기능 제공
 - ▶ vi의 기능과 사용법을 그대로 재 구현한 텍스트 편집기
 - ▶ 독자적으로 다양한 기능 추가
 - ▶ Vim 스크립트 등을 사용해서 자유롭게 편집 환경 변경 기능
 - ▶ 확장된 정규 표현식 문법 제공
 - ▶ 강력한 문법 강조 기능
 - ▶ 다중 되돌리기
 - ▶ 유니코드를 비롯한 다국어 지원
 - ▶ 문법 검사 지원
 - ▶ vi의 모든 기능을 제공하지는 않음



파일 편집

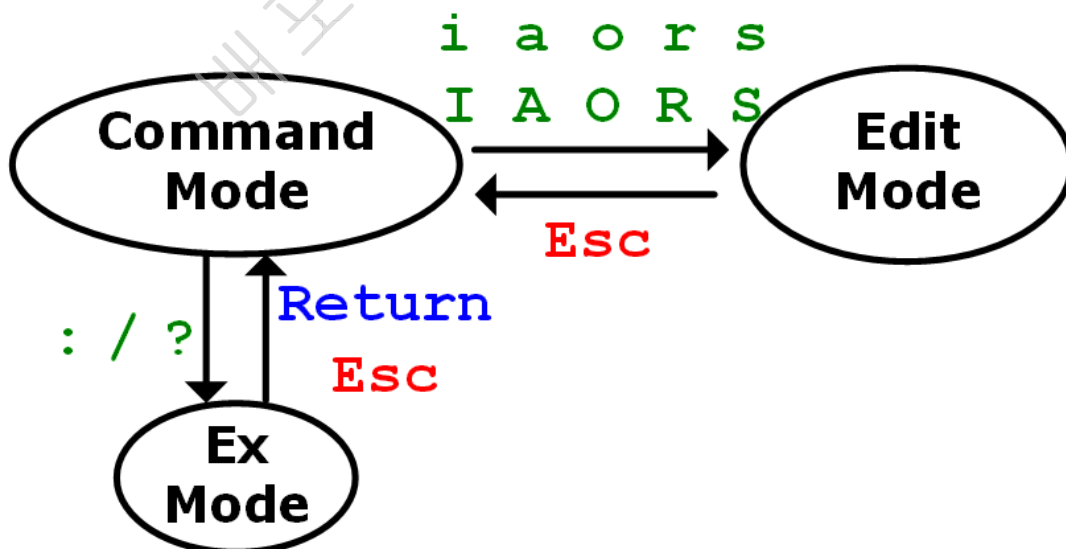
vi 사용하기

▶ 명령어 사용

- ▶ vi
 - ▶ 특정 파일 지정없이 vi 프로그램만 실행
- ▶ vi [파일명]
 - ▶ 명시된 파일이 없으면 생성하고 있으면 내용을 보여주며 실행
- ▶ 동작 모드의 이해 : 3가지
 - ▶ Command Mode
 - ▶ 디폴트 모드로 시작 시 이 모드 상태로 시작함
 - ▶ 삭제/복사/이동 등 편집 기능 등의 기능이 제공됨
 - ▶ Edit Mode
 - ▶ 텍스트 형태로 파일에 글자가 입력되는 상태로 일반적으로 메모장 상태라 간주하면 됨
 - ▶ Extended Mode
 - ▶ 확장 모드 상태로 텍스트 1줄 행태의 명령어를 조합하여 입력 후 엔터를 치면 기능 실행

파일 편집

vi 사용하기



파일 편집

vi 사용하기

▶ Command Mode to Edit Mode

- ▶ i : 삽입 모드 전환 + 현재 커서 위치에 글자 삽입
- ▶ I : 삽입 모드 전환 + 현재 커서가 위치한 줄 맨 앞에 위치에 글자 삽입
- ▶ a : 삽입 모드 전환 + 현재 커서 다음 위치에 글자 삽입
- ▶ A : 삽입 모드 전환 + 현재 커서가 위치한 줄 맨 마지막 위치에 글자 삽입
- ▶ o : 삽입 모드 전환 + 현재 커서가 위치한 줄 아래에 새로운 줄 생성 및 새 줄에서 글자 삽입
- ▶ O : 삽입 모드 전환 + 현재 커서가 위치한 줄 위에 새로운 줄 생성 및 새 줄에서 글자 삽입
- ▶ s : 삽입 모드 전환 + 현재 커서 위치의 글자 1개를 지우고 입력 모드로 전환
- ▶ S : 삽입 모드 전환 + 현재 커서가 위치한 줄을 모두 지우고 입력 모드로 전환
- ▶ r : 명령 모드 유지 + 현재 커서 글자에 1글자를 입력 받아 덮어씀.
- ▶ R : 현재 커서 위치부터 입력 받는 글자를 덮어씀

파일 편집

vi 사용하기

▶ Edit Mode to Command Mode

- ▶ ESC키 입력 시 커맨드 모드로 전환

▶ Command Mode to Extended Mode

- ▶ : (콜론) : 커서가 왼쪽 아래로 이동하며 명령어 라인 입력 상태로 전환
- ▶ / : 커서가 왼쪽 아래로 이동하며 검색 라인 입력 상태로 전환(커서 위치 기준 아래로 검색)
- ▶ ? : 커서가 왼쪽 아래로 이동하며 검색 라인 입력 상태로 전환(커서 위치 기준 위로 검색)

▶ Extended Mode to Command Mode

- ▶ ESC키 입력 시 커맨드 모드로 전환

파일 편집

vi 사용하기

▶ Command Mode 명령어

▶ 커서 이동

- ▶ 방향키 : 커서 이동
- ▶ h, l, j, k : 커서 이동, 방향키 역할(왼쪽, 오른쪽, 위, 아래)
- ▶ b, w : 이전 단어의 첫 글자로 / 다음 단어의 첫 글자로 이동
- ▶ B, W : 이전 단어의 첫 글자로 / 다음 단어의 첫 글자로 이동
- ▶ e : 다음 단어의 끝 글자로 이동
- ▶ E : 다음 단어의 끝 글자로 이동
- ▶ H, M, L : 화면의 맨 위 / 중간 / 맨 아래로
- ▶ O : 그 행의 맨 처음으로 (숫자 0)

- ▶ \$: 그 행의 맨 끝으로
- ▶ + : 다음 행의 처음으로
- ▶ - : 위 행의 처음으로
- ▶ 3| : 현재 행의 3번째 열로
- ▶ 4H : 화면 상의 처음 행부터 4행 밑으로
- ▶ 4L : 화면 상의 마지막 행부터 4행 위로
- ▶ (,) : 이전 문장의 시작/다음 문장의 시작
- ▶ {, } : 이전 문단의 시작/다음 문단의 시작
- ▶ [[,]] : 이전 섹션의 시작/다음 섹션의 시작

파일 편집

vi 사용하기

▶ Command Mode 명령어

▶ 행 이동

- ▶ 숫자 + G : 해당하는 행으로 이동
- ▶ G : 파일의 맨 끝 행으로 이동

▶ 삭제 명령

- ▶ x : 현재 커서 위치에 있는 한 글자 삭제
- ▶ X : 현재 커서 왼쪽의 한 글자 삭제
- ▶ dd : 한 행 잘라 내기(Ctrl + X 기능과 유사)
- ▶ 숫자+dd : 숫자만큼 행 잘라 내기(Ctrl + X 기능과 유사)
- ▶ D : 커서위치부터 오른쪽 끝까지 모두 잘라내기(Ctrl + X 기능과 유사)

▶ 내용의 복사 및 이동

- ▶ yy : 현재의 행 복사
- ▶ 숫자+yy : 현재의 행 위치부터 숫자 만큼 아래로 복사
- ▶ p : 아래(오른쪽)에 붙여넣기
- ▶ P : 위(왼쪽)에 붙여넣기

▶ 내용 고치기

- ▶ J : 현재 행과 아래 행 결합
- ▶ 3J : 3행 합치기
- ▶ u : 이전 명령 취소(Undo)
- ▶ Ctrl + r : 취소 명령 되돌리기(Redo)
- ▶ . : 마지막 명령 반복(상황에 따라 Redo용으로도 사용)
- ▶ 파일 저장 및 종료 : ZZ
- ▶ Visual Drag : v

파일 편집

vi 사용하기

▶ Extended Mode 명령어 [:(콜론)키]

- ▶ 행단위 이동
 - ▶ 숫자 : 해당하는 행으로 바로 이동
 - ▶ \$: 파일의 마지막 행으로 가기
- ▶ 파일 저장 및 종료
 - ▶ w : 저장
 - ▶ q : 종료
 - ▶ q! : 현재 수정했던 내용을 저장하지 않고 강제 종료
 - ▶ wq : 저장 후 종료
 - ▶ w [파일이름] : [파일이름]으로 저장 후 계속 편집

▶ 행 번호 설정 및 화면표시

- ▶ set nu : 행 번호 표시
- ▶ set nonu : 행 번호 숨기기
- ▶ = : 현재 행 번호 보여주기
- ▶ set noh : 문자열 검색 후 문자열 강조 끄기

▶ 치환

- ▶ %s/TTTT/tttt/g - 파일 전체(g)에서 'TTTT'를 'tttt'로 치환
 - ▶ % : 파일 전체를 의미(대신에 2, 10을 명시하면 2~10번 줄에 적용)
 - ▶ s(substitute) : 치환 명령
 - ▶ g(global) : 한 줄에 여러 개의 패턴이 있을 경우 모두 치환 (미사용시 처음 1개만 치환)
 - ▶ c(confirm) : 치환 요소 발생시 확인 받고 진행
 - ▶ i(ignore case) : 대소문자 구별없이 치환

파일 편집

vi 사용하기

▶ Extended Mode 명령어 [/키 혹은 ?키]

- ▶ 문자열 : 해당 문자열을 아래 혹은 위로 검색
- ▶ n : 찾던 방향으로 다음 문자열 검색
- ▶ N : 찾던 방향의 반대로 다음 문자열 검색

파일 편집

환경설정

▶ 환경변수 파일

- ▶ 홈 디렉터리에 설정 파일을 저장하여 vi가 실행할 때마다 참고하도록 할 수 있음
- ▶ 설정 파일명
 - ▶ vi : .exrc
 - ▶ vim : .vimrc
- ▶ Extended Mode에서 명령으로 환경 설정 가능
- ▶ abbr 혹은 ab 명령 : 약어/별칭 기능
- ▶ syntax 혹은 syn 명령 : 문법에 따른 강조 기능
 - ▶ on : 켜기
 - ▶ off : 끄기

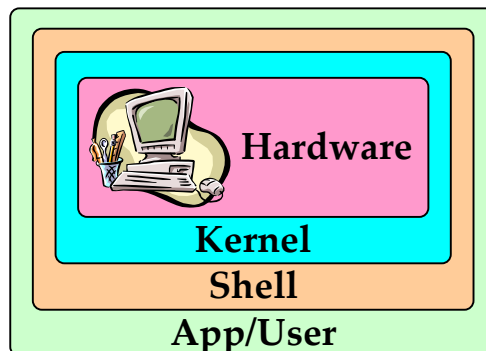
▶ set 명령

- ▶ autoindent 혹은 ai : 자동 들여쓰기
- ▶ smartindent : 특정 언어의 문법에 맞게 들여쓰기
- ▶ cindent : C프로그램 문법에 맞게 들여쓰기
- ▶ number 혹은 nu : 각 라인 앞에 라인 번호 출력
- ▶ showmatch : 괄호 계열 문자 입력 시 쌍을 찾아 주
- ▶ tabstop 혹은 n : 탭 간격 설정
- ▶ shiftwidth : 자동 들여쓰기 간격 설정
- ▶ ruler : 현재 커서 위치 표시

실습 : Shell

▶ 개념

- ▶ 사전적으로 껍질을 의미
- ▶ 운영체제 내부와 외부의 가교 역할
- ▶ 사용자의 명령어를 해석하고 번역
- ▶ 프로그래밍 기능 : 스크립트 형태로 가능
- ▶ 사용자의 이용 환경을 설정하는 기능



▶ 대표적 shell

- ▶ Thompson : 1971년 최초의 유닉스 셸. 켄 톰프슨이 작성
- ▶ Bourne : AT&T 벨 연구소의 스티븐 본이 개발
- ▶ C : 대화형 작업에 강점(스크립트가 C언어와 유사)
- ▶ Korn : Bourne 및 C의 좋은 특성들을 결합
- ▶ bash(Bourne Again shell) : Bourne 문법을 모두 만족. C shell 일부분도 포함

실습 : Shell

▶ 프롬프트

- ▶ Shell이 사용자의 명령을 대기하고 있는 모드
- ▶ 텍스트 형태로 표시

▶ Shell에 따른 프롬프트

| 셸 이름 | Bourne shell | Korn shell | BASH | C shell | Zsh | Tcsh |
|------|--------------|------------|-----------|----------|----------|-----------|
| 경로 | /bin/sh | /bin/ksh | /bin/bash | /bin/csh | /bin/zsh | /bin/tcsh |
| root | # | # | # | # | # | # |
| 사용자 | \$ | \$ | \$ | % | % | > |

실습 경로 이동

- ▶ 실습 1 : 로그인 후 현재 디렉터리 확인
- ▶ 실습 2 : 최상위 디렉터리로 이동(절대/상대 경로)
- ▶ 실습 3 : 실습2의 상태에서 "cd" 입력 후 엔터를 치고 pwd 명령으로 위치 확인
- ▶ 실습 4 : 실습2의 상태에서 "cd ~" 입력 후 엔터를 치고 pwd 명령으로 위치 확인
- ▶ 실습 5 : /usr/bin 디렉터리로 이동
- ▶ 실습 6 : 상대 경로를 통한 홈 디렉터리의 code 디렉터리로 이동
- ▶ 실습 7 : 실습 6의 상태에서 "cd -"를 입력 후 엔터를 치고 pwd 명령으로 위치 확인
- ▶ 실습 8 : 자동완성 기능 확인

실습

파일관리

▶ rm : Remove, 파일 제거 명령

- ▶ 용법 : rm [-옵션] 파일1 [파일2] [파일3] [파일4]...
- ▶ 주요 옵션
 - ▶ i : 확인 후 삭제
 - ▶ f : 강제 삭제
 - ▶ r : 디렉터리 삭제

▶ cp : Copy, 파일 복사 명령

- ▶ 용법 : cp [-옵션] 원본파일 복사파일
- ▶ 주요 옵션
 - ▶ r : 하위 디렉터리와 파일 모두 복사

실습

파일관리

▶ touch : 파일의 시간을 현재 시간으로 바꾸거나 새로운 파일 생성

- ▶ 용법 : touch [-옵션] 파일1 [파일2] [파일3] [파일4]...

▶ mv : Move, 파일을 이동하거나 이름을 바꿀 때 사용

- ▶ 용법 : mv [-옵션] 원본파일 이동파일

▶ cat : Catenate, 파일의 내용을 출력할 때 사용

- ▶ 용법 : cat [-옵션] 파일1 [파일2] [파일3] [파일4]...

▶ head : 파일의 앞 10행 출력

- ▶ 용법 : head [-옵션] 파일1 [파일2] [파일3] [파일4]...
- ▶ 주요 옵션
 - ▶ 숫자 : 출력 행 수 지정
 - ▶ c : 출력 용량 지정

실습

파일관리

- ▶ **tail** : 파일의 끝 10행 출력
 - ▶ 용법 : **tail** [-옵션] 파일1 [파일2] [파일3] [파일4]...
 - ▶ 주요 옵션
 - ▶ 숫자 : 출력 행 수 지정
 - ▶ c : 출력 용량 지정
- ▶ **clear** : 화면 지우기
- ▶ **more** : 파일의 내용을 화면단위로 보여줌
 - ▶ 용법 : **more** [-옵션] 파일1 [파일2] [파일3] [파일4]...
 - ▶ 실행 시 주요 키 사용법
 - ▶ q : 종료
 - ▶ space bar : 1페이지 이동 / 반대기능 : b
 - ▶ enter : 1 줄 이동
 - ▶ / : 검색

실습

파일관리

- ▶ *****(asterisk, 애스터리스크) : 모든 길이의 문자열에 대응하는 경우 사용
 - ▶ 실습 : 확장자가 txt인 모든 파일의 목록을 볼 때
 - ▶ ls *.txt
- ▶ **?** : 임의의 한문자
 - ▶ 실습 : 확장자가 txt이고 이름이 1글자인 파일의 목록을 볼 때
 - ▶ ls ?.txt
- ▶ **[]** : 대괄호, 괄호 문자로서 괄호 안 문자 중 하나를 포함하면 선택됨
 - ▶ 실습1 : 확장자가 txt이고, 이름이 1글자이며, BNK 중 1글자를 가진 파일의 목록을 볼 때
 - ▶ ls [BNK].txt
 - ▶ 실습2 : 확장자가 txt이고, 이름이 BNK2/BNK3/BNK4/BNK5인 파일의 목록을 볼 때
 - ▶ ls BNK[2-5].txt

실습

파일관리

▶ chmod : 파일의 접근/실행 권한을 수정할 수 있는 명령어(소유자/root 계정만 가능)

▶ 접근 권한의 숫자 표기

- ▶ r : 4
- ▶ w : 2
- ▶ x : 1

```
drwxrwxr-x
421 421 421
```

- ▶ rwxrwxr-x의 숫자 표기 : 775
- ▶ chmod 사용법1 : chmod 숫자표기 파일이름
- ▶ 예시) chmod 750 test.txt

▶ 접근 권한의 문자 표기

- ▶ u:사용자/g:그룹/o:기타 사용자/a:전체
- ▶ +:부여/+:제거로 r:읽기/w:쓰기/x:실행 권한을 수정
- ▶ chmod 사용법2 : chmod 문자표기 파일이름
- ▶ 예시) chmod ug+x test.txt

실습

셸 스크립트 기초

▶ echo : 글자 출력

- ▶ 용법 : echo [-옵션] [문자열]
- ▶ 실습
 - ▶ echo
 - ▶ echo 에코 테스트입니다.

▶ printf : 글자 출력

- ▶ 용법 : printf 문자열+[형식지정자] [인자1] [인자2] [인자3] ...
- ▶ 실습
 - ▶ printf 안녕하세요 printf 테스트 입니다.
 - ▶ printf "안녕하세요 printf 테스트 입니다."
 - ▶ printf "안녕하세요 printf 테스트 입니다.\n"
 - ▶ printf "100의 16진수 값은 %x입니다.\n" 100

실습

셸 스크립트 기초

▶ ;(세미 콜론)

- ▶ 여러 개의 명령을 1줄에 적을 때 사용
- ▶ 왼쪽부터 오른쪽 순으로 수행
- ▶ 실습
 - ▶ ls 와 pwd 결합

▶ |(파이프)

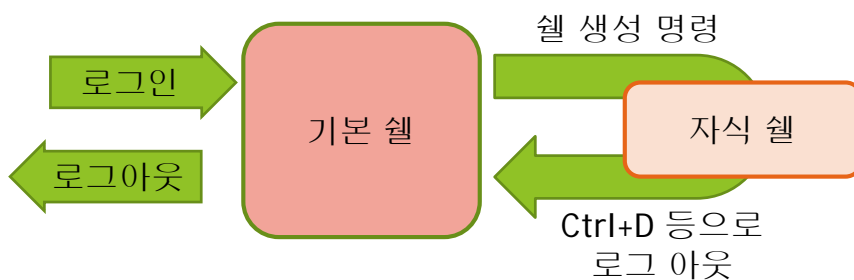
- ▶ 왼쪽의 실행 결과를 오른쪽에 전달할 때 사용
- ▶ 실습
 - ▶ ls -al과 head를 결합
 - ▶ ls -al과 tail을 결합

실습

셸 스크립트 기초

▶ 기본 셸과 자식 셸

- ▶ 기본 셸 : 로그인 할 때 수행되는 셸
- ▶ 자식 셸 : 로그인 후 기본 셸에서 추가 명령을 통해 생성되는 셸



실습

셸 스크립트 기초

- ▶ 변수
 - ▶ OS가 프로그램을 실행하기 위해 참조하는 기본적인 정보의 설정 값
 - ▶ 일반적으로 대문자를 사용(대소문자 구별)
- ▶ 환경변수 : 모든 셸에게 공통적으로 적용
- ▶ 셸 변수 : 자식 셸에게는 전파되지 않고 현재 셸에만 적용
- ▶ set 명령
 - ▶ 환경변수와 셸 변수 모두 출력
- ▶ env 명령
 - ▶ 환경변수만 출력

실습

셸 스크립트 기초

- ▶ 주요 환경 변수
 - ▶ PATH : 실행을 위한 명령어의 위치 정보
 - ▶ SHELL : 기본 셸 정보
 - ▶ HOME : 사용자 홈 디렉터리 경로
 - ▶ PWD : 현재 작업 디렉터리 경로
 - ▶ PS1 : 현재 프롬프트 설정 값
 - ▶ OSTYPE : 운영체제 타입
- ▶ 실습 : echo를 통한 환경 변수 확인
 - ▶ echo \$PWD
 - ▶ 다른 환경 변수도 확인

실습

셸 스크립트 기초

▶ 변수 생성 방법

▶ 환경변수

▶ 방법1

- ▶ 변수이름=값
- ▶ export 변수이름

▶ 방법2

- ▶ export 변수=값
- ▶ = 기호와 공백이 없어야 함

▶ 셸 변수

- ▶ 변수이름=값

▶ 변수 확인 방법 : echo 명령 활용

▶ 변수 제거 방법

- ▶ unset 변수이름

▶ 실습 : 환경변수 생성 및 출력1

▶ 실습1

- ▶ BNK_TEST="Hello BNK"
- ▶ echo \$BNK_TEST

▶ 실습2

- ▶ BNK_TEST="BNK Hello"
- ▶ echo \$BNK_TEST

실습

셸 스크립트 기초

▶ ' '와 " "의 문자열 감싸기 : 작은 따옴표/따옴표

- ▶ 문자열 내부에 사용된 특수 문자 기능을 해제
- ▶ " "의 경우 \$와 \는 특수 문자 기능을 해제하지 않음
- ▶ 실습

- ▶ echo '\$PWD'
- ▶ echo "\$PWD"

▶ ` `의 문자열 감싸기 : Backtick(백틱)

- ▶ ` `내용을 명령어로 간주하고 실행한 결과를 전달
- ▶ 실습
- ▶ echo "현재 디렉터리 리스트는 `ls -al` 입니다 "

실습

셸 스크립트 기초

▶ 실습 : 환경변수 생성 및 출력2

▶ 실습1

- ▶ `BNK_TEST=`ls -al``
- ▶ `echo $BNK_TEST`

▶ 실습2

- ▶ `BNK_TEST="ls -al"`
- ▶ `echo ` $BNK_TEST ``

실습

셸 스크립트 기초

▶ 리다이렉션

- ▶ 기호 : `>`, `>>`, `<`, `<<`
- ▶ 입출력의 방향을 전환시키는 역할
- ▶ 실습1 : 결과를 `a.txt` 파일을 생성한 다음 저장(기존 파일이 존재하면 삭제)
 - ▶ `ls -al > a.txt`
- ▶ 실습2 : 결과를 `a.txt` 파일을 생성한 다음 저장(기존 파일이 존재하면 끝에 추가)
 - ▶ `ls -al >> a.txt`

▶ 표준 입출력 장치 : 표준으로 사용되는 장치로 셸에서는 숫자를 부여해서 사용함

- ▶ `stdin` : Standard In, 데이터를 읽을 때 사용(키보드), 0번이 부여됨
- ▶ `stdout` : Standard Out, 데이터를 출력할 때 사용(화면), 1번이 부여됨
- ▶ `stderr` : Standard Error, 에러를 출력할 때 사용(화면), 2번이 부여됨

실습

셸 스크립트 기초

▶ 리다이렉션과 표준 입출력 장치 실습

▶ 실습1

- ▶ `tree > dir.txt`
- ▶ `cat dir.txt`

▶ 실습2

- ▶ `ls -zzz > a.txt`
- ▶ `cat a.txt`
- ▶ `ls -zzz 2> a.txt`
- ▶ `cat a.txt`

▶ 실습3 : 다음이 의미하는 것을 설명하시오

- ▶ `ls -al 2> a.txt 1>&2`

▶ 실습4 : 화면에 출력하지 않고 내용을 삭제하고 싶을 때

- ▶ `ls -al > /dev/null`

실습

셸 스크립트 기초

▶ alias : 별칭

- ▶ 특정 문자열을 다른 문자열로 대체하도록 설정하는 명령
- ▶ 사용법
 - ▶ `alias [별칭=문자열]`
 - ▶ `alias` 명령만 사용할 경우 설정된 값 출력
- ▶ 해제법
 - ▶ `unalias` 별칭

▶ history : 이전 명령 리스트 보기

- ▶ 이전 명령 리스트 모두 삭제 : `history -c`

▶ ! : 이전 명령 재 실행

- ▶ `!!` : 마지막 명령 재 수행
- ▶ `!번호` : `history`에 저장되어 있는 번호에 해당하는 명령 수행

실습

셸 스크립트 기초

- ▶ 프롬프트 변경
 - ▶ PS1의 변수 값 수정
 - ▶ PS1=문자열

실습

셸 스크립트 기초

- ▶ 스크립트 : 지껄이다
- ▶ 스크립트 언어 : 컴파일 단계 없이 바로 실행되는 컴퓨터 언어
 - ▶ 텍스트 형태로 되어 있고 결과를 바로 알 수 있어 사용이 편리함
 - ▶ HTML이나 Shell, java script 등이 대표적 스크립트 언어
- ▶ 셸 스크립트
 - ▶ 일반적으로 vi 편집기 등을 이용해서 작성
 - ▶ 실행 방법1
 - ▶ sh <스크립트 파일이름>
 - ▶ 실행 방법2
 - ▶ chmod로 실행 권한 부여 후 실행
 - ▶ chmod u+x <스크립트 파일이름>
 - ▶ ./<스크립트 파일이름>

실습

셸 스크립트 기초

▶ 셸 스크립트 구조

- ▶ 첫 줄은 아래의 텍스트로 시작
 - ▶ `#!/bin/sh`
 - ▶ 셸은 `#!` 다음 문자열을 스크립트의 실행 프로그램으로 해석
 - ▶ `/bin/sh` : 실행프로그램의 경로
- ▶ `#` : 스크립트에 삽입 시 이후부터 행 끝 까지를 주석으로 해석
- ▶ 첫 줄 이후 문법에 맞추어서 스크립트 작성

실습

셸 스크립트 기초 : 기본 셸 스크립트 작성

▶ HelloKitty.sh 작성 실습

1. `#!/bin/sh`
2. `#` 작성자 : 주수홍
3. `#` 기능 : 스크립트 테스트
- 4.
5. `echo "Hello Kitty" #` 지옥으로 키티 출력

bnk@VirtualBox: ~

k@VirtualBox:~\$ vi HelloKitty.sh

실습

셸 스크립트 기초 : 기본 셸 스크립트 작성

▶ HelloKitty.sh 작성 실습

```
bnk@VirtualBox: ~  
1 #!/bin/sh  
2 # 작성자 : 주수홍  
3 # 기능 : 스크립트 테스트  
4  
5 echo "Hello Kitty" # 지옥으로 키티 출력  
6  
~  
:wq  
bnk@VirtualBox:~$ vi HelloKitty.sh  
bnk@VirtualBox:~$ ll HelloKitty.sh  
-rw-rw-r-- 1 bnk bnk 121 HelloKitty.sh  
bnk@VirtualBox:~$ chmod 755 HelloKitty.sh  
bnk@VirtualBox:~$ ll HelloKitty.sh  
-rwxr-xr-x 1 bnk bnk 121 HelloKitty.sh  
bnk@VirtualBox:~$ ./HelloKitty.sh  
Hello Kitty  
bnk@VirtualBox:~$
```

실습

셸 스크립트 기초 : 변수 사용1

▶ HelloKitty2.sh 작성 실습

1. #!/bin/sh
2. # 변수 사용 스크립트
- 3.
4. BNK="Hello Kitty"
5. echo "BNK=\$BNK" # 지옥으로 키티 출력

```
bnk@VirtualBox: ~/code  
bnk@VirtualBox:~/code$ vi HelloKitty2.sh  
bnk@VirtualBox:~/code$ chmod 755 HelloKitty2.sh  
bnk@VirtualBox:~/code$ ./HelloKitty2.sh  
BNK=Hello Kitty  
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 기초 : 변수 사용2

▶ BNK.sh 작성 실습

1. `#!/bin/sh`
2. `# 변수 사용 스크립트`
- 3.
4. `y="2011"`
5. `m="3"`
6. `d="15"`
7. `echo "BNK금융그룹은 $y 년 $m 월 $d 일에 설립된 대한민국 최초의 지방은행 금융지주회사다."`

실습

셸 스크립트 기초 : 변수 사용2

▶ BNK.sh 작성 실습

```
bnk@VirtualBox: ~/code
1 #!/bin/sh
2 # 변수 사용 스크립트
3
4 y="2011"
5 m="3"
6 d="15"
7 echo "BNK금융그룹은 $y 년 $m 월 $d 일에 설립된 대한민국 최초의 지방은행 금융지주회사다."
8
```

8,0-1

```
bnk@VirtualBox: ~/code
bnk@VirtualBox:~/code$ vi BNK.sh
bnk@VirtualBox:~/code$ chmod 755 BNK.sh
bnk@VirtualBox:~/code$ ./BNK.sh
BNK금융그룹은 2011 년 3 월 15 일에 설립된 대한민국 최초의 지방은행 금융지주회사다.
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 기초 : 변수 사용3

▶ BNK2.sh 작성 실습

1. #!/bin/sh
2. # 변수 사용 스크립트
- 3.
4. y="2011"
5. m="3"
6. d="15"
7. echo "BNK금융그룹은 \${y}년 \${m}월 \${d}일에 설립된 대한민국 최초의 지방은행 금융지주회사다."

실습

셸 스크립트 기초 : 변수 사용3

▶ BNK2.sh 작성 실습

```
bnk@VirtualBox: ~/code
1 #!/bin/sh
2 # 변수 사용 스크립트
3
4 y="2011"
5 m="3"
6 d="15"
7 echo "BNK금융그룹은 ${y}년 ${m}월 ${d}일에 설립된 대한민국 최초의 지방은행 금융지주회사다."
8
~
:wq
```

```
bnk@VirtualBox: ~/code
bnk@VirtualBox:~/code$ ./BNK2.sh
BNK금융그룹은 2011년 3월 15일에 설립된 대한민국 최초의 지방은행 금융지주회사다.
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 기초 : 제어문

▶ 프로그램의 흐름을 바꾸는 문법

▶ if문 구조

```
if [ 조건 ]  
then  
    코드  
fi
```

▶ if문 특징

- ▶ 조건문 괄호안의 단어 사이에는 **반드시 공백**이 있어야 함
- ▶ 조건의 결과가 참이면 코드를 수행
- ▶ 비교연산자 =, != 사용 가능

▶ if문 조건(파일 관련)

- ▶ -d 파일이름 : 디렉터리면 참
- ▶ -e 파일이름 : 파일이 존재하면 참
- ▶ -f 파일이름 : 일반 파일이면 참
- ▶ -r 파일이름 : 읽기 가능하면 참
- ▶ -w 파일이름 : 쓰기 가능하면 참
- ▶ -x 파일이름 : 실행 가능하면 참
- ▶ -s 파일이름 : 크기가 0이 아니면 참
- ▶ 조건문 예시1) [-d "BNK.sh"]
- ▶ 조건문 예시2) [-e "BNK.sh"]

실습

셸 스크립트 기초 : 제어문

▶ if_test.sh 작성 실습

1. #!/bin/sh
2. if ["\$SHELL" = "/bin/bash"]
3. then
4. echo "당신의 셸은 본 셸입니다."
5. fi

```
bnk@VirtualBox: ~/code  
bnk@VirtualBox:~/code$ ./if_test.sh  
당신의 셸은 본 셸입니다.  
bnk@VirtualBox:~/code$
```


실습

셸 스크립트 기초 : 제어문

▶ 프로그램의 흐름을 바꾸는 문법

▶ if-else문 구조

```
if [ 조건 ]  
then  
    코드1  
else  
    코드2  
fi
```

```
if [ 조건 ] ; then  
    코드1  
else  
    코드2  
fi
```

▶ if문 특징

- ▶ 조건의 결과가 참이면 코드1, 거짓이면 코드2 수행

▶ if문 조건(비교연산자)

- ▶ "문자열1" = "문자열2" : 같으면 참
- ▶ "문자열1" != "문자열2" : 다르면 참
- ▶ -n "문자열" : null이 아니면 참
- ▶ -z "문자열" : null이면 참

▶ if문 조건(산술 비교연산자)

- ▶ 수식1 -eq 수식2 : 같으면 참
- ▶ 수식1 -ne 수식2 : 다르면 참
- ▶ !수식 : 수식이 거짓이면 참
- ▶ 수식1 -gt 수식2 : 수식1 > 수식2이면 참
- ▶ 수식1 -ge 수식2 : 수식1 >= 수식2이면 참
- ▶ 수식1 -lt 수식2 : 수식1 < 수식2이면 참
- ▶ 수식1 -le 수식2 : 수식1 <= 수식2이면 참

실습

셸 스크립트 기초 : 제어문

▶ if_test2.sh 작성 실습

1. #!/bin/sh
2. if ["\$SHELL" = "/bin/bash"]
3. then
4. echo "당신의 셸은 본 셸입니다."
5. else
6. echo "당신의 셸은 본 셸이 아닙니다."
7. fi

```
bnk@VirtualBox: ~/code  
1 #!/bin/sh  
2 if [ "$SHELL" = "/bin/bash" ]  
3 then  
4 echo "당신의 셸은 본 셸입니다."  
5 else  
6 echo "당신의 셸은 본 셸이 아닙니다."  
7 fi  
8
```

```
bnk@VirtualBox: ~/code  
bnk@VirtualBox:~/code$ ./if_test2.sh  
당신의 셸은 본 셸입니다.  
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 기초 : 제어문

- ▶ 실습1 : if-else문을 사용하여 숫자 1과 10이 같은 지 비교하시오
- ▶ 실습2 : if-else문을 사용하여 "BNK"와 "bnk"가 같은 지 비교하시오
- ▶ 실습3 : if-else문을 사용하여 ~/code가 디렉터리인지 확인하시오
- ▶ 실습4 : if-else문을 사용하여 /etc/passwd를 읽기 가능한지 확인하시오
- ▶ 실습5 : if-else문을 사용하여 /bin/lis에 쓰기 가능한지 확인하시오
- ▶ 실습6 : if-else문을 사용하여 ~/.vimrc가 일반 파일인지 확인하시오
- ▶ 실습7 : if-else문을 사용하여 /bin/cat이 실행 가능한지 확인하시오

실습

셸 스크립트 기초 : 인자값 입력 처리

- ▶ 인자값 : 스크립트 실행시 명령 라인의 문자열
 - ▶ 예시)
 - ▶ ./BNK.sh param1 param2 param3 param4
 - ▶ \$0 \$1 \$2 \$3 \$4
 - ▶ 예시의 효과 : 아래와 같은 변수를 선언한 것으로 간주 됨
 - ▶ \$0=./BNK.sh
 - ▶ \$1=param1
 - ▶ \$2=param2
 - ▶ \$3=param3
 - ▶ \$4=param4

실습

셸 스크립트 기초 : 인자값 입력 처리

▶ param_test.sh 작성 실습

1. `#!/bin/sh`
2. `echo "인자0 = ${0}"`
3. `echo "인자1 = ${1}"`
4. `echo "인자2 = ${2}"`
5. `echo "인자3 = ${3}"`
6. `echo "인자4 = ${4}"`

```
bnk@VirtualBox: ~/code
bnk@VirtualBox:~/code$ ./param_test.sh
인 자 0 = ./param_test.sh
인 자 1 =
인 자 2 =
인 자 3 =
인 자 4 =
bnk@VirtualBox:~/code$ ./param_test.sh 1111 22222 333333 444444
인 자 0 = ./param_test.sh
인 자 1 = 1111
인 자 2 = 22222
인 자 3 = 333333
인 자 4 = 444444
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 문법과 활용 : 키보드 입력

▶ read문

- ▶ 키보드 입력을 변수에 대입
- ▶ 구조

```
read 변수명
```

▶ read_test.sh 작성 실습

1. `#!/bin/sh`
2. `echo "글자를 입력하십시오"`
3. `read INPUT_DATA`
4. `echo "입력된 글자는 ${INPUT_DATA}입니다."`

```
bnk@VirtualBox: ~/code
bnk@VirtualBox:~/code$ ./read_test.sh
글 자 를   입 력 하 시 오
양   호   랑   이   다
입 력 된 글 자 는   [양   호   랑   이   다   ]입 니 다 .
bnk@VirtualBox:~/code$
```

실습

셸 스크립트 문법과 활용 : 제어문2

▶ case in esac문

- ▶ 여러 사례에 단순한 구조로 대응할 수 있음
- ▶ 구조

| |
|---|
| case 문자열 in |
| 비교문자열1) 명령1;; |
| 비교문자열2-1 비교문자열2-2 ...) 명령2-1 |
| ... |
| 명령2-n;; |
| 비교문자열3) 명령3;; |
| ... |
| *) 명령4;; |
| esac |

실습

셸 스크립트 문법과 활용 : 제어문2

▶ case_test1.sh 작성 실습

1. #!/bin/sh
2. case "\$1" in
3. 일)
4. echo "1";;
5. 1)
6. echo "일";;
7. *)
8. echo "글쎄";;
9. esac

```
bnk@VirtualBox: ~/code
1 #!/bin/sh
2 case "$1" in
3    일 )
4        echo "1";;
5    1 )
6        echo "일 ";;
7    * )
8        echo "글 쎄 ";;
9 esac
10
```

```
bnk@VirtualBox: ~/code
bnk@VirtualBox:~/code$ ./case_test1.sh 1
bnk@VirtualBox:~/code$ ./case_test1.sh 일
bnk@VirtualBox:~/code$ ./case_test1.sh 글 쎄
bnk@VirtualBox:~/code$
```

실습

크립트 문법과 활용 : 제어문2

▶ case_test2.sh 작성 실습

```
1. #!/bin/sh
2. echo "1~9 숫자 입력"
3. read NUMBER
4. case "$NUMBER" in
5.     1|2|3)
6.         echo "123"
7.         ;;
8.     4|5|6)
9.         echo "456"
10.        ;;
11.    7|8|9)
12.        echo "789"
13.        ;;
14.    *)
15.        echo "입력 오류"
16.        ;;
17. esac
```

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 echo "1~9 숫자 입력"
3 read NUMBER
4 case "$NUMBER" in
5     1|2|3)
6         echo "123"
7         ;;
8     4|5|6)
9         echo "456"
10        ;;
11    7|8|9)
12        echo "789"
13        ;;
14    *)
15        echo "입력 오류"
16        ;;
17 esac
18
```

```
nk@VirtualBox: ~/code
VirtualBox:~/code$ sh case_test2.sh
숫자 입력

VirtualBox:~/code$ sh case_test2.sh
숫자 입력

VirtualBox:~/code$ sh case_test2.sh
숫자 입력

오류
VirtualBox:~/code$
```

실습

셸 스크립트 문법과 활용 : 숫자 연산

▶ expr : Evaluate Expressions

- ▶ 정수 연산 수행용 명령어
- ▶ 연산자와 숫자 사이는 반드시 공백 삽입
- ▶ 숫자 연산 문자열을 명령과 사용시 반드시 `(백틱)으로 묶어 사용
- ▶ <, >, |, &, * 의 경우 백슬래시(\)를 붙여서 사용 : \<, \>, \|, \&, *
- ▶ 괄호 사용시 백슬래시(\)를 붙여서 사용
- ▶ 사례1 : echo `expr 3 + 3`
- ▶ 사례2 : echo `expr 3 * 3`
- ▶ 사례3 : echo `expr 3 \> 3`
- ▶ 사례4 : echo `expr \(3 + 3 \) * 4`
- ▶ 실습 : (3+2)*(5-2) 수식을 표현하시오

실습

셸 스크립트 문법과 활용 : 숫자 연산

▶ bc : Basic Calculator

- ▶ 실수 연산도 가능하며 일반적인 수식으로 계산 가능
- ▶ 파이프(|)를 통해 수식 문자열을 전달하여 계산
 - ▶ 사례) `echo "(3+2)*(5-2)" | bc`
- ▶ 소수점 자리 수를 지정하지 않으면 정수 값만 출력 : `scale`로 지정
 - ▶ 사례) `echo "2/3"|bc`
 - ▶ 사례) `echo "scale=2;2/3"|bc`
- ▶ 진법을 지정하여 계산 가능 : `ibase/obase`
 - ▶ 사례) `echo "ibase=16;A+B"|bc`
 - ▶ 사례) `echo "obase=2;2+3"|bc`

실습

셸 스크립트 문법과 활용 : 숫자 연산

▶ expr.sh 작성 실습

1. `#!/bin/sh`
2. `NUM1=1`
3. `NUM2=2`
4. `NUM3=`expr ${NUM1} + ${NUM2}``
5. `NUM4=`expr ${NUM1}+${NUM2}``
6. `echo ${NUM3}`
7. `echo ${NUM4}`

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 NUM1=1
3 NUM2=2
4 NUM3=`expr ${NUM1} + ${NUM2}`
5 NUM4=`expr ${NUM1}+${NUM2}`
6 echo ${NUM3}
7 echo ${NUM4}
8
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh expr.sh
3
1+2
bnk@VirtualBox:~$
```


실습

셸 스크립트 문법과 활용 : 숫자 연산

▶ bc.sh 작성 실습

1. `#!/bin/sh`
2. `NUM1=10`
3. `NUM2=2`
4. `NUM3=`echo "sqrt(${NUM1}^${NUM2})" | bc``
5. `NUM4=`echo "3.2+5.1" | bc``
6. `echo ${NUM3}`
7. `echo "3.2+5.1=${NUM4}"`

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 NUM1=10
3 NUM2=2
4 NUM3=`echo "sqrt(${NUM1}^${NUM2})" | bc`
5 NUM4=`echo "3.2+5.1" | bc`
6 echo ${NUM3}
7 echo "3.2+5.1=${NUM4}"
8
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh bc.sh
10
3.2+5.1=8.3
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : AND와 OR 연산자

▶ AND : `&&` 혹은 `-a` 사용

▶ OR : `||` 혹은 `-o` 사용

▶ andor.sh 작성 실습

1. `#!/bin/sh`
2. `echo "파일명을 입력하시오"`
3. `read DATA1`
4. `echo =====`
5. `if [-e ${DATA1}] && [-r ${DATA1}] ; then`
6. `head -3 ${DATA1}`
7. `else`
8. `echo "${DATA1}파일을 읽을 수 없습니다"`
9. `fi`
10. `echo =====`
11. `if [-e ${DATA1} -a -r ${DATA1}] ; then`
12. `head -3 ${DATA1}`
13. `else`
14. `echo "${DATA1}파일을 읽을 수 없습니다"`
15. `fi`
16. `echo =====`

실습

셸 스크립트 문법과 활용 : AND와 OR 연산자

▶ andor.sh 작성 실습

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 echo "파 일 명 을 입 력 하 시 오 "
3 read DATA1
4 echo =====
5 if [ -e ${DATA1} ] && [ -r ${DATA1} ] ; then
6     head -3 ${DATA1}
7 else
8     echo "${DATA1}파 일 을 읽 을 수 없 습 니 다 "
9 fi
10 echo =====
11 if [ -e ${DATA1} -a -r ${DATA1} ] ; then
12     head -3 ${DATA1}
13 else
14     echo "${DATA1}파 일 을 읽 을 수 없 습 니 다 "
15 fi
16 echo =====
17
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh andor.sh
파 일 명 을 입 력 하 시 오
andor.sh
=====
#!/bin/bash
echo "파 일 명 을 입 력 하 시 오 "
read DATA1
=====
#!/bin/bash
echo "파 일 명 을 입 력 하 시 오 "
read DATA1
=====
bnk@VirtualBox:~$
bnk@VirtualBox:~$ sh andor.sh
파 일 명 을 입 력 하 시 오
test.tt
=====
test.tt파 일 을 읽 을 수 없 습 니 다
=====
test.tt파 일 을 읽 을 수 없 습 니 다
=====
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 셸 변수2

- ▶ \${변수명:-String}
 - ▶ 변수명이 존재하지 않으면 String값 사용
- ▶ \${변수명:=String}
 - ▶ 변수명이 존재하지 않거나 널이면 String을 대입
- ▶ \${변수명:+String}
 - ▶ 변수명이 존재하고 값이 널이 아니면 String을 대입
- ▶ \${변수명:?String}
 - ▶ 변수명이 존재하고 널이 아니면 그 값을 사용
 - ▶ 위 조건을 만족하지 않으면 String을 출력하고 종료

실습

셸 스크립트 문법과 활용 : 셸 변수2

▶ val.sh 작성 실습

```
1. #!/bin/sh
2. DATA1="BNK GROUP"
3. echo "1: $DATA1"
4. echo "2: ${DATA1:-"헬로키티2"}"
5. echo "3: ${DATA2:-"헬로키티3"}"
6. echo "4: ${DATA1:= "헬로키티4"}"
7. echo "5: ${DATA2:= "헬로키티5"}"
8. echo "6: ${DATA2:= "헬로키티6"}"
9. echo "7: ${DATA1:+ "헬로키티7"}"
10. echo "8: ${DATA1:? "헬로키티8"}"
11. echo "9: ${DATA3:? "헬로키티9"}"
12. echo =====
```

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 DATA1="BNK GROUP"
3 echo "1: $DATA1"
4 echo "2: ${DATA1:-"헬로키티2"}"
5 echo "3: ${DATA2:-"헬로키티3"}"
6 echo "4: ${DATA1:= "헬로키티4"}"
7 echo "5: ${DATA2:= "헬로키티5"}"
8 echo "6: ${DATA2:= "헬로키티6"}"
9 echo "7: ${DATA1:+ "헬로키티7"}"
10 echo "8: ${DATA1:? "헬로키티8"}"
11 echo "9: ${DATA3:? "헬로키티9"}"
12 echo =====
13
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh val.sh
1: BNK GROUP
2: BNK GROUP
3: 헬로키티3
4: BNK GROUP
5: 헬로키티5
6: 헬로키티5
7: 헬로키티7
8: BNK GROUP
val.sh: 11: val.sh: DATA3: 헬로키티
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 문자열 제거

- ▶ #, % : 문자열을 검색해서 첫번째 일치하는 문자열 제거
 - ▶ #은 앞에서 검색 %는 뒤에서 검색
 - ▶ ##, %% : 쉘 앞과 쉘 끝을 찾아 사이에 존재하는 스트링을 모두 제거

▶ string_test.sh 작성 실습

```
1. #!/bin/sh
2. PATH_VAL="/home/bnk/code/home/bnk/code/bnk/code"
3. echo ${PATH_VAL%/bnk*}
4. echo ${PATH_VAL%%/bnk*}
```

실습

셸 스크립트 문법과 활용 : 문자열 제거

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 PATH_VAL="/home/bnk/code/home/bnk/code/bnk/code"
3 echo ${PATH_VAL%/bnk*}
4 echo ${PATH_VAL%%/bnk*}
5
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh string_test.sh
/home/bnk/code/home/bnk/code
/home
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 문자열 길이

▶ # : 변수가 가지고 있는 문자열의 길이를 반환

▶ count_test.sh 작성 실습

```
1. #!/bin/sh
2. DATA1="123456789"
3. DATA2="1234567"
4. LEN1=${#DATA1}
5. LEN2=${#DATA2}
6. echo "DATA1 = ${LEN1}"
7. echo "DATA2 = ${LEN2}"
```

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 DATA1="123456789"
3 DATA2="1234567"
4 LEN1=${#DATA1}
5 LEN2=${#DATA2}
6 echo "DATA1 = ${LEN1}"
7 echo "DATA2 = ${LEN2}"
8
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh count_test.s
DATA1 = 9
DATA2 = 7
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 명령행의 인자

| 명령행의 인자 | 설명 |
|----------------|-------------------|
| \$0 | 셸 스크립트 이름 |
| \$1 ~ \$9 | 1번부터 9번까지의 인자 |
| \${10}~\${100} | 10번째 인자~100번째 인자 |
| \$# | 전체 인자 개수 |
| \$* 혹은 \$@ | 모든 인자 |
| "\$*" | "\$1 \$2 \$3" |
| "\$@" | "\$1" "\$2" "\$3" |
| \$? | 수행된 명령의 종료 값 |

실습

셸 스크립트 문법과 활용 : 명령행의 인자

▶ cmdline.sh 작성 실습

1. `#!/bin/sh`
2. `echo '$*' : '$*`
3. `echo '$#' : '$#`
4. `echo '$@' : '$@`
5. `echo " : $1 $2 $3"`

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 echo '$*' : '$*'
3 echo '$#' : '$#'
4 echo '$@' : '$@'
5 echo " : $1 $2 $3"
6
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh cmdline_test.sh 일 이 삼
$* : 일 이 삼
$# : 3
$@ : 일 이 삼
: 일 이 삼
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 반복문

▶ for ~ in 문 구조

```
for 변수 in list
do
    명령
done
```

- ▶ list에 나열된 값을 변수에 대입하며 list의 개수 만큼 명령을 반복

실습

셸 스크립트 문법과 활용 : 반복문

▶ forin_test.sh 작성 실습

1. #!/bin/sh
2. for VAL in 0 1 2
3. do
4. echo "VAL = \${VAL}"
5. done

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 for VAL in 0 1 2
3 do
4     echo "VAL = ${VAL}"
5 done
6
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh forin_test.sh
VAL = 0
VAL = 1
VAL = 2
bnk@VirtualBox:~$
```

실습

셸 스크립트 문법과 활용 : 반복문

▶ forin_test2.sh 작성 실습

1. #!/bin/sh
2. for VAL in \$*
3. do
4. echo "Arg = [\$VAL]"
5. done

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2
3 for VAL in $*
4 do
5     echo "Arg = [$VAL]"
6 done
7 █
```

```
bnk@VirtualBox: ~
bnk@VirtualBox:~$ sh forin_test2.sh 원 투 쓰 리
Arg = [원 ]
Arg = [투 ]
Arg = [쓰 리 ]
bnk@VirtualBox:~$ █
```

실습

셸 스크립트 문법과 활용 : 반복문

- ▶ 실습1 : 구구단 중 5단을 출력하시오
- ▶ 실습2 : 구구단 2단~9단을 모두 출력하시오

실습

셸 스크립트 문법과 활용 : 반복문

▶ while 문 구조

```
while [ 조건 ]  
do  
    명령  
done
```

▶ 조건이 참인 동안 명령을 반복

▶ while_test.sh 작성 실습

```
1. #!/bin/sh  
2. COUNT=1  
3. SUM=0  
4.  
5. while [ $COUNT -le 10 ]  
6. do  
7.     SUM=`expr ${SUM} + ${COUNT}`  
8.     COUNT=`expr ${COUNT} + 1`  
9. done  
10.  
11. echo "1부터 10까지 합 : $SUM"
```

실습

셸 스크립트 문법과 활용 : 반복문

bnk@ozsystem: ~/code/old

```
1 #!/bin/sh  
2 COUNT=1  
3 SUM=0  
4  
5 while [ $COUNT -le 10 ]  
6 do  
7     SUM=`expr ${SUM} + ${COUNT}`  
8     COUNT=`expr ${COUNT} + 1`  
9 done  
10  
11 echo "1부터 10까지 합 : $SUM"  
12
```

bnk@ozsystem: ~/code/old

```
bnk@ozsystem:~/code/old$ sh while_test.sh  
1부터 10까지 합 : 55  
bnk@ozsystem:~/code/old$
```

실습

셸 스크립트 문법과 활용 : 반복문

- ▶ 실습1 : "BNK GROUP " 값을 가진 PASS변수를 만들고 키보드로 문자열을 입력 받아 일치할 때까지 무한 반복하는 스크립트를 작성하시오
- ▶ 실습2 : 구구단 2단~9단을 while문을 사용해 모두 출력하시오

실습

셸 스크립트 문법과 활용 : 반복문

▶ until 문 구조

```
until [ 조건 ]  
do  
    명령  
done
```

- ▶ 조건이 거짓인 동안 명령을 반복

▶ until_test.sh 작성 실습

```
1. #!/bin/sh  
2. COUNT=1  
3. SUM=0  
4.  
5. until [ $COUNT -gt 10 ]  
6. do  
7.     SUM=`expr ${SUM} + ${COUNT}`  
8.     COUNT=`expr ${COUNT} + 1`  
9. done  
10.  
11. echo "1부터 10까지 합 : $SUM"
```

실습

셸 스크립트 문법과 활용 : 반복문

▶ until_test.sh 작성 실습

```
bnk@ozsystem: ~/code/old
1 #!/bin/sh
2 COUNT=1
3 SUM=0
4
5 while [ $COUNT -le 10 ]
6 do
7     SUM=`expr ${SUM} + ${COUNT}`
8     COUNT=`expr ${COUNT} + 1`
9 done
10
11 echo "1부 터  10까 지  합  : $SUM"
12
```

```
bnk@ozsystem: ~/code/old
bnk@ozsystem:~/code/old$ sh until_test.sh
1부 터  10까 지  합  : 55
bnk@ozsystem:~/code/old$
```

실습

셸 스크립트 문법과 활용 : 프로그램의 종료 값

▶ ? 변수

- ▶ 최근에 종료된 프로그램의 종료 값이 저장되어 있는 변수
- ▶ 0은 정상 종료를 나타내며 그 외의 값은 비정상 종료를 나타냄
- ▶ echo로 확인 가능
- ▶ 사용 사례
 - ▶ ls
 - ▶ echo \$?
 - ▶ ls -zzzz
 - ▶ echo \$?

실습

셸 스크립트 문법과 활용 : 셸의 종료 값

▶ exit : 셸 종료 키워드

- ▶ 셸의 실행을 종료 시킴
- ▶ 로그인 한 상황에서 사용시 로그 아웃의 역할을 수행
- ▶ 셸 스크립트에 사용 시 스크립트가 바로 종료됨
- ▶ 셸 스크립트에 사용 시 키워드 다음에 정수 값을 적시
 - ▶ 사례1) exit 0
 - ▶ 사례2) exit 1
 - ▶ 사례2) exit 2
 - ▶ 0은 정상 종료, 이외의 값은 비정상 종료를 나타냄

실습

셸 스크립트 문법과 활용 : 반복문

▶ select - in 문 구조(배서 셸만 가능)

```
select 변수 in list
do
    명령
done
```

- ▶ list에 나열된 문자열로 자동으로 메뉴 생성
- ▶ 선택된 메뉴를 변수에 대입

▶ PS 변수

- ▶ PS1 : 기본 프롬프트 변수
- ▶ PS2 : 보조 프롬프트 변수
 - ▶ 기본값 : >
 - ▶ "\"를 사용하여 명령 행을 연장시 표시됨
- ▶ PS3 : 셸 스크립트의 select문 구동시 사용되는 프롬프트 변수
 - ▶ 기본값 : #?
- ▶ PS4 : 셸 스크립트 디버깅 모드의 프롬프트 변수
 - ▶ 기본값 : +

실습

셸 스크립트 문법과 활용 : 반복문

▶ select_test.sh 작성 실습

```
1. #!/bin/bash
2. PS3="번호를 입력하세요 : "
3. select CHOICE in "Date" "Tree" "Exit"
4. do
5.     case ${CHOICE} in
6.         "Tree")
7.             tree
8.             ;;
9.         "Date")
10.            date
11.            ;;
12.        "Exit")
13.            exit 0
14.            ;;
15.        *)
16.            echo "입력 오류"
17.            ;;
18.    esac
19. done
```

실습

셸 스크립트 문법과 활용 : 반복문

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 PS3="번호를 입력하세요 : "
4
5 select CHOICE in "Date" "Tree" "Exit"
6 do
7     case ${CHOICE} in
8         "Tree")
9             tree
10            ;;
11        "Date")
12            date
13            ;;
14        "Exit")
15            exit 0
16            ;;
17        *)
18            echo "입력 오류"
19            ;;
20    esac
21 done
22
```

```
bnk@ozsystem: ~/code/test
bnk@ozsystem:~/code/test$ ./select_test.sh
1) Date
2) Tree
3) Exit
번호를 입력하세요 : 1
2019. KST
번호를 입력하세요 : 2
$ select_test.sh
0 directories, 1 file
번호를 입력하세요 : 3
bnk@ozsystem:~/code/test$
```

실습

셸 스크립트 기초 : 제어문 3

▶ if - elif 문 구조

```
if [ 조건 ]  
then  
    코드  
elif [ 조건 ]  
then  
    코드  
fi
```

▶ if - elif -else 문 구조

```
if [ 조건 ]  
then  
    코드  
elif [ 조건 ]  
then  
    코드  
else  
    코드  
fi
```

실습

셸 스크립트 문법과 활용 : 반복문

▶ select_test2.sh 작성 실습

```
1.  #!/bin/bash  
2.  
3.  PS3="대한민국 최초의 지방은행 금융지주회사는?"  
4.  CORRECT="BNK"  
5.  EXIT="모름"  
6.  
7.  select CHOICE in "BMW" "KIA" "${CORRECT}" "${EXIT}"  
8.  do  
9.      if [ ${CHOICE} = "${CORRECT}" ]  
10.     then  
11.         echo "BNK금융그룹은 대한민국 최초의 지방은행 금융지주회  
12.         exit 0  
13.     elif [ ${CHOICE} = "${EXIT}" ]  
14.     then  
15.         echo "종료합니다."  
16.         exit 0  
17.     else  
18.         echo "[${CHOICE}] 는 틀렸습니다."  
19.     fi  
20. done
```

실습

셸 스크립트 문법과 활용 : 반복문

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 PS3="대한민국 최초의 지방은행 금융지주회사는 ? "
4 CORRECT="BNK"
5 EXIT="모름"
6
7 select CHOICE in "BMW" "KIA" "${CORRECT}" "${EXIT}"
8 do
9     if [ ${CHOICE} = "${CORRECT}" ]
10    then
11        echo "BNK금융그룹은 대한민국 최초의 지방은행 금융지주회사다"
12        exit 0
13    elif [ ${CHOICE} = "${EXIT}" ]
14    then
15        echo "종료합니다."
16        exit 0
17    else
18        echo "[${CHOICE}] 는 틀렸습니다."
19    fi
20 done
21
```

실습

셸 스크립트 문법과 활용 : 반복문

```
bnk@ozsystem: ~/code
bnk@ozsystem:~/code$ ./select_test2.sh
1) BMW
2) KIA
3) BNK
4) 모름
대한민국 최초의 지방은행 금융지주회사는 ? 1
[BMW] 는 틀렸습니다.
대한민국 최초의 지방은행 금융지주회사는 ? 2
[KIA] 는 틀렸습니다.
대한민국 최초의 지방은행 금융지주회사는 ? 3
BNK금융그룹은 대한민국 최초의 지방은행 금융지주회사다
bnk@ozsystem:~/code$ ./select_test2.sh
1) BMW
2) KIA
3) BNK
4) 모름
대한민국 최초의 지방은행 금융지주회사는 ? 4
종료합니다.
bnk@ozsystem:~/code$
```


실습

셸 스크립트 문법과 활용 : 반복문

▶ select_test3.sh 작성 실습

```
1. #!/bin/bash
2.
3. PS3="번호를 입력하세요: "
4.
5. select CHOICE in ls pwd date exit
6. do
7.     ${CHOICE}
8. done
```

bnk@ozsystem: ~/code

```
1 #!/bin/bash
2
3 PS3="번호를 입력하세요: "
4
5 select CHOICE in ls pwd date
6 do
7     ${CHOICE}
8 done
9 █
```

bnk@ozsystem: ~/code

```
bnk@ozsystem:~/code$ ./select_test3.sh
1) ls
2) pwd
3) date
4) exit
번호를 입력하세요 : 1
old passwd.sh select test.sh select test2.sh select te
번호를 입력하세요 : 2
/home/bnk/code
번호를 입력하세요 : 3
2019. KST
번호를 입력하세요 : 4
bnk@ozsystem:~/code$ █
```

실습

셸 스크립트 연계 실습 및 기타 : 반복문

▶ continue 문

- ▶ 반복문 안에서 사용
- ▶ 이 키워드를 만나면 프로그램의 흐름이 반복문의 처음으로 이동

▶ break 문

- ▶ 반복문 안에서 사용
- ▶ 이 키워드를 만나면 반복문이 즉시 종료됨

▶ passwd2.sh 작성 실습(read -p는 bash에서 동작)

```
1. #!/bin/bash
2.
3. while [ 1 ]
4. do
5.     read -p "비밀번호를 입력하세요: "
6.     if [ "${REPLY}" != "BNK GROUP" ]
7.     then
8.         continue
9.     fi
10.    break
11. done
12. echo "비밀번호를 맞게 입력하셨습니다."
```

실습

셸 스크립트 연계 실습 및 기타 : 반복문

bnk@ozsystem: ~/code

```
1 #!/bin/bash
2
3 while [ 1 ]
4 do
5     read -p "비밀번호를 입력하세요 : "
6     if [ "${REPLY}" != "BNK GROUP" ]
7     then
8         continue
9     fi
10    break
11 done
12 echo "비밀번호를 맞게 입력하셨습니다 ."
13
```

bnk@ozsystem: ~/code

```
bnk@ozsystem:~/code$ ./passwd2.sh
비밀번호를 입력하세요 : 1
비밀번호를 입력하세요 : BNK GROUP
비밀번호를 맞게 입력하셨습니다 .
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : 문자열 입력 활용

bnk@ozsystem: ~/code

```
bnk@ozsystem:~/code$ ./istring.sh
실행할까요? (y/n)y
실행됨
bnk@ozsystem:~/code$ ./istring.sh
실행할까요? (y/n)n
실행 취소됨
bnk@ozsystem:~/code$
```

bnk@ozsystem: ~/code

```
1 #!/bin/bash
2
3 read -p "실행할까요? (y/n)" CHOICE
4
5 if [[ $CHOICE = [yY]* ]]
6 then
7     echo "실행됨"
8 else
9     echo "실행 취소됨"
10 fi
11
```

▶ istring.sh 작성 실습

1. #!/bin/bash
- 2.
3. read -p "실행할까요? (y/n)" CHOICE
- 4.
5. if [[\$CHOICE = [yY]*]]
6. then
7. echo "실행됨"
8. else
9. echo "실행 취소됨"
10. fi

실습

셸 스크립트 연계 실습 및 기타 : 함수

- ▶ 목적을 달성하기 위한 관련된 명령어로 만들어진 그룹
- ▶ 구조

```
function 함수이름
{
    명령
    return
}
```

- ▶ 내부에 return을 만나면 종료함
- ▶ return 다음에 값을 적을 경우 ?변수에 저장됨 : \$?으로 값의 활용 가능
- ▶ 인자 전달 가능
 - ▶ 함수 호출시 함수이름 다음에 값을 적을 경우 1, 2, 3 등의 숫자 변수에 저장됨
 - ▶ \$1, \$2, \$3 등으로 활용 가능

실습

셸 스크립트 연계 실습 및 기타 : 함수

- ▶ func_test.sh 작성 실습

```
1. #!/bin/bash
2.
3. function Normal
4. {
5.     echo "Normal call"
6.     return
7. }
8.
9. Normal
```

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 function Normal
4 {
5     echo "Normal call"
6     return
7 }
8
9 Normal
10
```

```
bnk@ozsystem: ~/code
bnk@ozsystem:~/code$ ./func_test
Normal call
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : 함수

▶ func_test2.sh 작성 실습

```
1. #!/bin/bash
2.
3. function RetValue
4. {
5.     echo "RetValue"
6.     return 100
7. }
8.
9. RetValue
10. echo "함수 반환값[${?}]"
```

bnk@ozsystem: ~/code

```
1 #!/bin/bash
2
3 function RetValue
4 {
5     echo "RetValue"
6     return 100
7 }
8
9 RetValue
10 echo "함수 반환값[${?}]"
11
```

bnk@ozsystem: ~/code

```
bnk@ozsystem:~/code$ ./func_test2.sh
RetValue
함수 반환값 [100]
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : 함수

▶ func_test3.sh 작성 실습

```
1. #!/bin/bash
2.
3. function ParamTest
4. {
5.     echo "ParamTest : ${1}, ${2}"
6.     return `expr ${1} + ${2}`
7. }
8.
9. ParamTest 30 40
10. echo "함수 반환값[${?}]"
```

bnk@ozsystem: ~/code

```
1 #!/bin/bash
2
3 function ParamTest
4 {
5     echo "ParamTest : ${1}, ${2}"
6     return `expr ${1} + ${2}`
7 }
8
9 ParamTest 30 40
10 echo "함수 반환값[${?}]"
11
```

bnk@ozsystem: ~/code

```
bnk@ozsystem:~/code$ ./func_test3.sh
ParamTest : 30, 40
함수 반환값 [70]
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : eval

▶ 문자열을 명령어로 인지하고 실행

▶ eval_test.sh 작성 실습

```
1. #!/bin/sh
2.
3. COMMAND="ls -al"
4.
5. echo "=====
6. echo ${COMMAND}
7. echo "=====
8. eval ${COMMAND}
```

```
bnk@ozsystem: ~/code/test
1 #!/bin/sh
2
3 COMMAND="ls -al"
4
5 echo "=====
6 echo ${COMMAND}
7 echo "=====
8 eval ${COMMAND}
9
```

```
bnk@ozsystem: ~/code/test
bnk@ozsystem:~/code/test$ ./eval_test.sh
=====
ls -al
=====
합 계 12
drwxrwxr-x 2 bnk bnk 4096  8월  22  02:06 .
drwxrwxr-x 4 bnk bnk 4096  8월  22  02:04 ..
-rwxr-xr-x 1 bnk bnk  132  8월  22  02:05 eval_test
bnk@ozsystem:~/code/test$
```

실습

셸 스크립트 연계 실습 및 기타 : set

▶ set_test.sh 작성 실습

```
1. #!/bin/bash
2.
3. set $(date)
4.
5. for VAL in $@
6. do
7.     echo $VAL
8. done
```

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 set $(date)
4
5 for VAL in $@
6 do
7     echo $VAL
8 done
9
```

```
bnk@ozsystem: ~/code
bnk@ozsystem:~/code$ ./set_test.sh
2019.
01.
10.
(목 )
19:38:18
KST
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : 디버깅

- ▶ `bash -x` 셸스크립트 이름 : 각 행이 실행될 때 마다 출력
- ▶ `trap` : 실행중인 스크립트에서 특정 변수 값 추적
- ▶ `set_test.sh` 작성 실습

```
1. #!/bin/bash
2.
3. set $(date)
4. trap 'echo "$LINENO : VAL=$VAL"' DEBUG
5. for VAL in $@
6. do
7.     echo $VAL
8. done
```

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 set $(date)
4 trap 'echo "$LINENO : VAL=$VAL"' DEBUG
5 for VAL in $@
6 do
7     echo $VAL
8 done
9
```

```
bnk@ozsystem: ~/code
bnk@ozsystem:~/code$ ./trap_test
5 : VAL=
7 : VAL=2019.
2019.
5 : VAL=2019.
7 : VAL=01.
01.
5 : VAL=01.
7 : VAL=10.
10.
5 : VAL=10.
7 : VAL=(목 )
(목 )
5 : VAL=(목 )
7 : VAL=12:51:31
12:51:31
5 : VAL=12:51:31
7 : VAL=KST
KST
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : shift

- ▶ 파라미터 변수를 하나씩 삭제하며 이동 시킴
- ▶ `shfit_test.sh` 작성 실습

```
1. #!/bin/bash
2.
3. set $(date)
4.
5. while [ "$1" != "" ]
6. do
7.     echo $1
8.     shift
9. done
```

```
bnk@ozsystem: ~/code
1 #!/bin/bash
2
3 set $(date)
4
5 while [ "$1" != "" ]
6 do
7     echo $1
8     shift
9 done
10
```

```
bnk@ozsystem: ~/code
bnk@ozsystem:~/code$ ./shift_test.
2019.
01.
10.
(목 )
13:02:31
KST
bnk@ozsystem:~/code$
```

실습

셸 스크립트 연계 실습 및 기타 : printf

- ▶ 형식화된 출력이 가능한 **bash**의 내장 명령
- ▶ " " 안에 형식 지정자 %로 출력 값의 모양을 지정할 수 있음
- ▶ 주요 형식 지정자
 - ▶ %d : 10진수(%10d : 10자리를 확보한 후 숫자 출력, %04d : 4자리 확보 후 빈자리는 0으로 채움)
 - ▶ %s : 문자열
 - ▶ %f : 실수(%11.9f : 소수점 이하 9자리만 출력, 11자리를 확보한 후 숫자 출력)
 - ▶ %x : 16진수
- ▶ 주요 특수 기능
 - ▶ \t : 탭 삽입
 - ▶ \n : 엔터 삽입

실습

셸 스크립트 연계 실습 및 기타 : let

- ▶ **expr**보다 간단한 연산 가능 수행
- ▶ **let_test.sh** 작성 실습

```
bnk@ozsystem:~$ ./let_test.sh
13 + 7 = 20
20 + 7 = 27
16 / 4 = 4
4 - 5 = -1
7 * 10 = 70
70을 8로 나눈 나머지는 = 6
```

```
1. #!/bin/bash
2.
3. let NUM1=13
4. let NUM1=NUM1+7
5. echo "13 + 7 = $NUM1"
6. let "NUM1 = NUM1+7"
7. echo "20 + 7 = $NUM1"
8.
9. let NUM2=16
10. let "NUM2 /= 4"
11. echo "16 / 4 = $NUM2"
12. let "NUM2 -= 5"
13. echo "4 - 5 = $NUM2"
14.
15. let NUM3=7
16. let "NUM3 = NUM3 * 10"
17. echo "7 * 10 = $NUM3"
18. let "NUM3 %= 8"      # let "NUM3 = NUM3
19. echo "70을 8로 나눈 나머지는 = $NUM3"
```


실습

셸 스크립트 연계 실습 및 기타 : declare/typeset

- ▶ 변수의 특성을 설정할 때 사용
- ▶ 용법 : declare/typeset -옵션 변수명
 - ▶ -r : 읽기 전용 변수 생성, 저장하려면 오류 발생
 - ▶ -i : 변수를 정수로만 사용, 문자열을 대입할 경우 0이 됨
 - ▶ \$ declare -i NUM
 - ▶ \$ NUM =3
 - ▶ \$ echo "NUM = \$ NUM"
 - ▶ \$ NUM =three
 - ▶ \$ echo "NUM = \$ NUM"
 - ▶ -x : 셸 변수를 환경 변수로 만들어 서브 셸에서 사용할 수 있게 됨
 - ▶ -f : 스크립트안의 함수 목록 출력

실습

셸 스크립트 연계 실습 및 기타 : exec

- ▶ 현재 스크립트를 종료시키고 새로운 프로세스를 생성하여 명령 수행
- ▶ exec_test.sh 작성 실습

1. #!/bin/bash
- 2.
3. exec echo \$(pwd)
4. # 다음 줄은 실행 안됨
5. echo "=====

```
bnk@ozsystem:~$ ./exec_test.sh
/home/bnk
bnk@ozsystem:~$
```

실습

셸 스크립트 연계 실습 및 기타 : 날짜/시간 명령어

- ▶ **date** : 날짜와 시간을 출력
- ▶ **zdump** : 타임존에 해당하는 시간 출력
 - ▶ 형식 : **zdump** 타임존
 - ▶ 타임존 : KST(한국), PST(태평양), HST(하와이)
- ▶ **time/times** : 명령어 실행 시간에 대한 통계 출력
 - ▶ 형식 : **time** 명령어
 - ▶ 사례) **time ls -al**
- ▶ **cal** : 달력 출력
- ▶ **sleep** : 초 단위 대기
- ▶ **usleep** : 마이크로 초 단위 대기

실습

셸 스크립트 연계 실습 및 기타 : 파일 관련

- ▶ **diff** : 파일을 비교(디렉터리 비교도 가능)
 - ▶ 형식 : **diff** [-옵션] 파일1 파일2
 - ▶ 옵션 **u** : 파일의 차이를 새로운 형식으로 통합하여 보여 줌(시간 -/+ 등)
 - ▶ 종료 정보를 같으면 0, 다르면 1을 남김으로 셸 스크립트에 활용됨
 - ▶ **diff3** : 파일 3개 비교
 - ▶ **cmp** : 간단한 비교 가능(파일의 다른 부분만 출력, **diff**와 동일한 종료 정보)
- ▶ **patch**
 - ▶ **diff**로 생성된 파일의 차이점에 근거하여 파일을 수정
 - ▶ New File - Old File = **patch** File(**diff**로 생성)
 - ▶ **diff -Naur new/main.c old/main.c > main.patch**
 - ▶ Old File + **patch** File(**patch**로 적용) = New File
 - ▶ **patch -p0 < main.patch**

patch file : a/arch/x86/mm/fault.c

-p0 : a/arch/x86/mm/fault.c
-p1 : arch/x86/mm/fault.c
-p2 : x86/mm/fault.c
-p3 : mm/fault.c

실습

셸 스크립트 연계 실습 및 기타 : 파일 관련

▶ **tar** : 유닉스 표준 아카이브 유틸리티

- ▶ **tar** -옵션 tar파일명 압축대상파일들
- ▶ **-c** : 묶기
- ▶ **-x** : 풀기
- ▶ **-z** : 압축 옵션 적용
- ▶ **-v** : 처리 과정 표시
- ▶ **-f** : 파일이름지정
- ▶ 사례1) `tar -cvf test.tar *`
- ▶ 사례2) `tar -xzvf test.tar.gz`

▶ **gzip** : GNU zip, 파일 1개를 압축

- ▶ 사례1) `gzip test.tar`
- ▶ 사례2) `gzip -d test.tar.gz`

실습

셸 스크립트 연계 실습 및 기타 : 파일 관련

▶ **basename** : 문자열에서 경로 정보를 삭제하고 파일 이름만 추출

- ▶ 사례) `basename `pwd``

▶ **dirname** : 문자열에서 파일 이름을 삭제하고 경로만 추출

- ▶ 사례) `dirname `pwd``

실습

셸 스크립트 연계 실습 및 기타 : 통신 관련

- ▶ nslookup : 특정 서버의 인터넷 정보 출력
 - ▶ 사례) nslookup google.com
- ▶ traceroute : 특정 서버까지의 경로 탐색
- ▶ ping : 네트워크 상태 체크용 유틸
 - ▶ 성공시 종료상태를 0으로 남김으로 스크립트에서 활용 가능
 - ▶ 사례) ping google.com
- ▶ sx/rx : xmodem 프로토콜, 파일 전송 유틸
- ▶ sz/rz : zmodem 프로토콜, 파일 전송 유틸, xmodem 보다 기능이 뛰어남
- ▶ ssh : 보안이 적용된 텍스트형 원격 셸 프로그램
- ▶ rcp/rsh/telnet : 보안이 적용되지 않은 복사/셸 원격 프로그램

실습

셸 스크립트 연계 실습 및 기타 : awk

- ▶ 문자열과 관련된 다양한 기능을 제공하는 GNU 프로젝트로 만들어진 명령어
- ▶ 제작자 : Aho, Weinberger, Kernighan
- ▶ 형식
 - ▶ awk '패턴' 파일이름
 - ▶ awk '{행위}' 파일이름
 - ▶ awk '패턴 {행위}' 파일이름
- ▶ 사례
 - ▶ ls -al | awk '{print \$9}'
 - ▶ ls -al | awk '/\..sh/'
 - ▶ ls -al | awk '/\..sh/{print"\t찾은 파일은 " \$9 "이며 " \$3 " 소유입니다."}'
 - ▶ ls -al | awk '\$5 < 1000'
 - ▶ ls -al | awk '\$5 < 1000{print \$9}'

실습

셸 스크립트 연계 실습 및 기타 : sed

- ▶ 문자열과 관련된 다양한 기능을 제공
- ▶ 사례
 - ▶ `ls -al | sed -n '/\.sh/p'`
 - ▶ `ls -al | sed 's/\.sh/ooooooooooooo/'`
 - ▶ `ls -al | sed 's/ \./-/'`
 - ▶ `ls -al | sed '\./!d'`

실습

셸 스크립트 연계 실습 및 기타 : 기타

- ▶ `seq` : 일정한 정수를 생성
 - ▶ 사례1) `seq 10`
 - ▶ 사례2) `seq 10 15`
 - ▶ 사례3) `seq 10 3 20`
 - ▶ 사례4) `for DAN in `seq 2 9``
 - ▶ `for`문을 이용하여 구구단을 작성하시오
- ▶ `banner` : 큰 글자 생성(한글은 지원 안됨)
- ▶ `tee` : 명령 전달 파이프 중간에 삽입되어 출력의 내용을 복사하는 유틸
 - ▶ 사례) `ls -alR | tee a.txt | more`
- ▶ `dexdump` : 파일의 내용을 16진수로 출력

실습

셸 스크립트 연계 실습 및 기타 : 관리자 명령들

- ▶ `chown, chgrp` : 파일/디렉터리의 소유주와 소유 그룹 변경
- ▶ `useradd, userdel` : 사용자 생성 및 삭제
 - ▶ `adduser`는 `useradd`에 편의를 추가한 스크립트로 작성되어 있음
- ▶ `wall` : 전체 사용자에게 실시간 메시지 전송
- ▶ `dmesg` : 커널이 출력한 메시지 표시
- ▶ `uname` : 시스템 정보 출력
- ▶ `arch` : 시스템 아키텍처 출력
- ▶ `free` : 메모리 상황 출력
- ▶ `procinfo` : 프로세서 상황을 출력
- ▶ `du` : 현재 디렉터리와 그 하부 파일들의 사용량 출력
- ▶ `df` : 파일시스템별 사용량 출력

실습

셸 스크립트 연계 실습 및 기타 : 관리자 명령들

- ▶ `netstat` : 네트워크 상태 출력
- ▶ `uptime` : 부팅 후 서버 러닝 시간 출력
- ▶ `hostname` : 서버명 출력
- ▶ `ps` : 현재 셸에서 실행 중인 프로세스 출력
 - ▶ `ps -a` : 자신의 ID로 운영 중인 모든 프로세스 출력
 - ▶ `ps -A` : 시스템 전체의 프로세스 출력
- ▶ `pstree` : 실행 중인 프로세스를 트리 구조로 출력
- ▶ `top` : 실시간 cpu 사용 통계 출력
- ▶ `init`

실습

셸 스크립트 연계 실습 및 기타 : init와 런레벨

- ▶ init : 런레벨 설정
- ▶ 런레벨 : OS체제의 동작 모드
- ▶ 런레벨 확인 : `who -r`(BSD 제외)
 - ▶ 리눅스의 런레벨
 - ▶ 0 : Halt
 - ▶ 1 : Single user mode
 - ▶ 2 : Multiuser Mode(CLI, Not Networking)
 - ▶ 3 : Multiuser Mode(CLI)
 - ▶ 4 : Unused
 - ▶ 5 - Multiuser Mode(GUI)
 - ▶ 6 - Reboot
 - ▶ 솔라리스의 런레벨
 - ▶ 0 : PROM 모드
 - ▶ S : Single user mode(파일 시스템 사용 불가)
 - ▶ 1 : Single user mode
 - ▶ 2 : Multiuser Mode(NFS 클라이언트 모드)
 - ▶ 3 : Multiuser Mode(NFS 서버 모드)
 - ▶ 4 : Unused
 - ▶ 5 - Power Off
 - ▶ 6 - Reboot