

Linux 환경에서 개발 환경 구축 및 활용

1. 개요

본 문서는 Linux 환경에서 C/C++ 언어를 이용하여 개발을 시작하려는 개발자를 위한 교육을 목적으로 작성되었습니다.

본 문서에서 다루게 될 내용은 Linux OS 설치, Linux 주요 명령어, C/C++ 개발 환경 설치 및 설정, C/C++을 이용한 App 개발시 유용한 Tool의 사용법, 기타 유용한 명령어 및 Utility의 사용법 등을 다루게 됩니다.

2. Linux 소개

Linux는 Unix 운영 체제를 기반으로 하는 오픈 소스 운영 체제입니다. 서버, 데스크톱 컴퓨터 및 모바일 장치에 널리 사용됩니다. Linux는 안정성, 보안 및 유연성으로 유명하여 다양한 유형의 환경에서 널리 사용됩니다.

Linux 환경에서 시스템은 **커널, 시스템 라이브러리 및 사용자 공간 프로그램**을 포함하여 여러 부분으로 나뉩니다. 커널은 운영 체제의 핵심이며 메모리 및 처리 능력과 같은 시스템 리소스를 관리합니다. 시스템 라이브러리는 사용자 공간 프로그램이 파일 읽기 및 쓰기와 같은 일반적인 작업을 수행하는 데 사용할 수 있는 일련의 기능을 제공합니다. 사용자 공간 프로그램은 운영 체제 위에서 실행되고 사용자가 상호 작용하는 기능을 제공하는 응용 프로그램입니다.

Linux는 또한 모든 파일과 디렉토리가 루트 디렉토리에서 시작하여 트리와 같은 구조로 구성되는 계층적 파일 시스템을 사용합니다. 이렇게 하면 시스템에서 파일을 쉽게 탐색하고 찾을 수 있습니다.

각각 고유한 기능과 도구 세트가 있는 다양한 Linux 배포판이 있습니다. 일부 인기 있는 배포판에는 **Ubuntu, Debian 및 Red Hat**이 있습니다. 각 배포판에는 시스템에 소프트웨어를 설치하고 업데이트하는 데 도움이 되는 자체 패키지 관리자가 있을 수 있습니다.

요약하면 **Linux는 서버, 데스크톱 컴퓨터 및 모바일 장치에 널리 사용되는 강력하고 유연하며 안전한 운영 체제입니다.** 다양한 환경의 요구 사항에 맞게 사용자 정의할 수 있는 많은 기능과 도구를 제공합니다.

2.1 Linux 장/단점

2.1.1 장점

1. 유닉스와 완벽하게 호환이 가능 (POSIX를 따른다고 함)
2. 공개된 운영체제이며 무료임
3. 하드웨어의 기능이 파일 구조로 쉽게 접근이 가능하게 되어있음
4. 강력한 네트워크가 구축되어 있음
5. 보안기능
6. 오픈소스임
7. 다중사용자, 다중 작업, 대화형 인터페이스이기 때문에 서버에 적합

2.1.2 단점

1. 공개이다보니 책임이 따르지 않음
2. 기술지원이 빠르지 않다
3. 사용자의 숙련된 기술과 경험이 중요함

POSIX ?

POSIX(Portable Operating System interface) 의 약자로, 서로 다른 UNIX OS 의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 만든 인터페이스 규격임
커널로서 C 인터페이스 시스템 콜, 프로세스 환경, 파일과 디렉토리, 시스템 DB, tar 압축 포맷 등 다양한 분야를 이루고 있음

2.2 Linux 배포판 종류

Linux 는 크게 3가지의 배포판이 존재 합니다. 각 배포판은 유사한 명령어 및 Utility 를 사용 할 수 있습니다.

예를 들어 `package manager` 를 CentOS 에서는 yum 을 사용하고 Ubuntu 에서는 apt 를 사용하는 등 각 배포판별 다를 app 을 사용하고 있습니다.

- 데비안 (Debian) 계열 주요 Linux 배포판
 - Debian, Kali Linux, **Ubuntu**, Linux Mint, ...
- 레드햇(RedHat) 계열 주요 Linux 배포판
 - RedHat, Fedora, **CentOS**, CentOS 후속 **Rocky Linux** (로키 리눅스), ...
- 슬랙웨어(Slackware) 계열 주요 Linux 배포판
 - **Open SuSE**, Vector Linux, 쉘릭스(Salix) OS, ...

2.3 Linux 주요 directory 소개

Linux 시스템에서 파일 시스템은 '/'로 표시되는 루트 디렉토리에서 시작하는 계층 구조로 구성됩니다. 특정 용도를 제공하는 Linux 파일 시스템 내에는 몇 가지 중요한 디렉토리가 있습니다.

- **/bin**: 이 디렉토리에는 명령줄 유틸리티와 같이 시스템이 작동하는 데 필수적인 이진 실행 파일이 포함되어 있습니다.
- **/sbin**: 이 디렉토리에는 시스템 관리 유틸리티와 같은 시스템 바이너리 실행 파일이 포함되어 있습니다.
- **/dev**: 각종 디바이스 파일들이 위치해 있는데 크게 블록 디바이스와 캐릭터 디바이스로 나눌 수 있다. **블록 디바이스**란 HDD와 같은 주변 장치를 말하는데, 데이터가 블록 단위로 읽고 쓰여지며 랜덤하게 액세스할 수 있다. 반면 **캐릭터 디바이스**는 입출력이 한 바이트 단위로 이루어지며 데이터가 순차적으로 읽고 쓰여진다. 디바이스를 새로 만들 때에는 **mknod** 명령을 이용하면 되며, 물론 **/bin**에 위치해 있다. mount를 할 때에 필요한 디바이스 몇 개만 소개한다.
 - **/dev/fd0**: 플로피 디스크 디바이스
 - 첫 번째 FDD의 디바이스로 두 번째의 경우는 0 대신 1을, 세 번째의 경우는 2를 써주면 된다.
 - **/dev/hda or /dev/hda1**: IDE 하드 디스크 디바이스

- 마지막 부분의 hda에서 'a'는 위치를 나타내는 것으로, 'a'는 primary master, 'b'는 primary slave, 'c'는 secondary master, 'd'는 secondary slave를 의미한다. 그리고 그 뒤에 숫자가 없을 경우에는 전체를 의미한다. 숫자를 달 경우에는 파티션을 의미한다.
- **/dev/sda, /dev/sda1**: SCSI 하드 디스크 디바이스
 - IDE 하드 디스크 디바이스와 같으나 'h' 대신 's'를 쓴다.
- **/dev/cdrom**: CD-ROM 디바이스이다
- **/etc**: 이 디렉토리에는 시스템 및 설치된 소프트웨어에 대한 구성 파일이 포함되어 있습니다.
- **/usr**: 이 디렉토리에는 라이브러리, 문서 및 응용 프로그램과 같은 사용자 관련 파일이 포함되어 있습니다.
- **/var**: 이 디렉토리에는 로그 파일, 스푼 파일 및 임시 파일과 같은 가변 데이터가 들어 있습니다.
- **/lib**: 이 디렉토리에는 프로그램이 일반적인 작업을 수행하는 데 사용하는 코드 모음인 시스템 라이브러리가 포함되어 있습니다.
- **/opt**: 이 디렉토리는 기본 시스템의 일부가 아닌 추가 소프트웨어를 설치하는 데 사용됩니다.
- **/home**: 이 디렉토리에는 개인 파일과 설정을 저장할 수 있는 시스템의 모든 사용자 홈 디렉토리가 포함되어 있습니다.
- **/root**: 이 디렉토리는 시스템에 대한 모든 액세스 및 권한을 가진 슈퍼유저인 루트 사용자의 홈 디렉토리입니다.
- **/tmp**: 이 디렉토리는 프로그램이나 시스템에서 생성한 임시 파일을 저장하는 데 사용됩니다.
 - /tmp directory 에 저장되어 있는 파일은 OS 가 재 시작 되면 저장되어 있던 파일이 삭제됩니다.

이러한 각 디렉토리는 특정 용도로 사용되며 시스템이 제대로 작동하는 데 필수적인 파일을 포함합니다. 이러한 디렉토리와 여기에 포함된 파일의 목적을 이해하면 Linux 시스템 문제를 해결하고 관리할 때 도움이 될 수 있습니다.

3. Linux 설치 환경 소개

환경에 따라 Linux 를 설치하는 방법은 몇 가지로 나뉩니다.

- PC 에 USB Device 또는 CD 를 이용하여 설치하는 방법
- VMWare Workspation Player 를 이용하여 설치하는 방법(추천)
- VirtualBox 를 이용하여 설치하는 방법
- **WSL** 을 이용하는 설치하는 방법(추천)
- Docker 를 이용하는 방법 (VirtualBox 가 설치되어 있어야 한다)
- 클라우드 환경을 이용하는 방법
 - AWS
 - 네이버 클라우드
 - 가비아

4. Linux 설치

우선 별첨된 [VMware Workstation Player 15 다운로드, 설치 및 사용 방법.md](#) 문서를 이용하여 VMware Workstation Player 를 설치 합니다.

문서를 작성하는 시점에 Wmware Workstation Player(이하 VMware) 의 최신 Version 은 17 입니다.

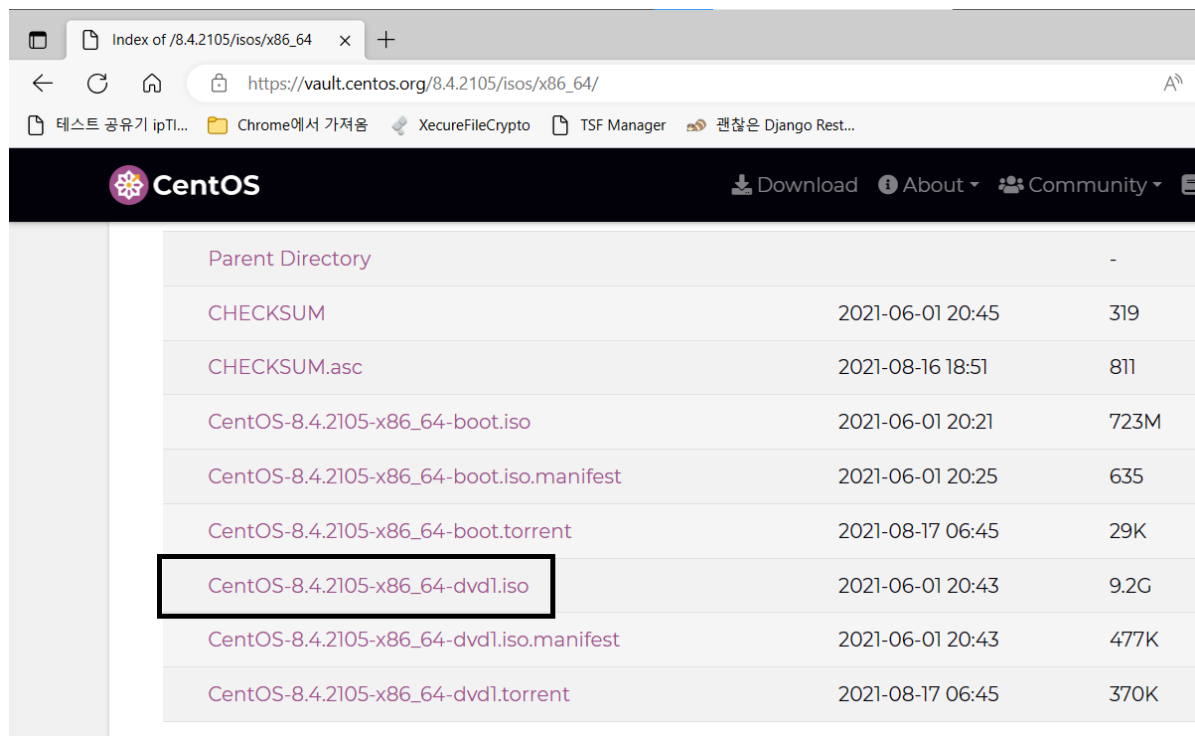
<https://www.vmware.com/kr/products/workstation-player/workstation-player-evaluation.html>

4.1 VMWare 를 이용하여 CentOS 8.4 를 설치하는 방법

문서를 작성하는 시점에 대부분의 [Mirror](#) 에 CentOS ISO Image 가 존재하지 않습니다.

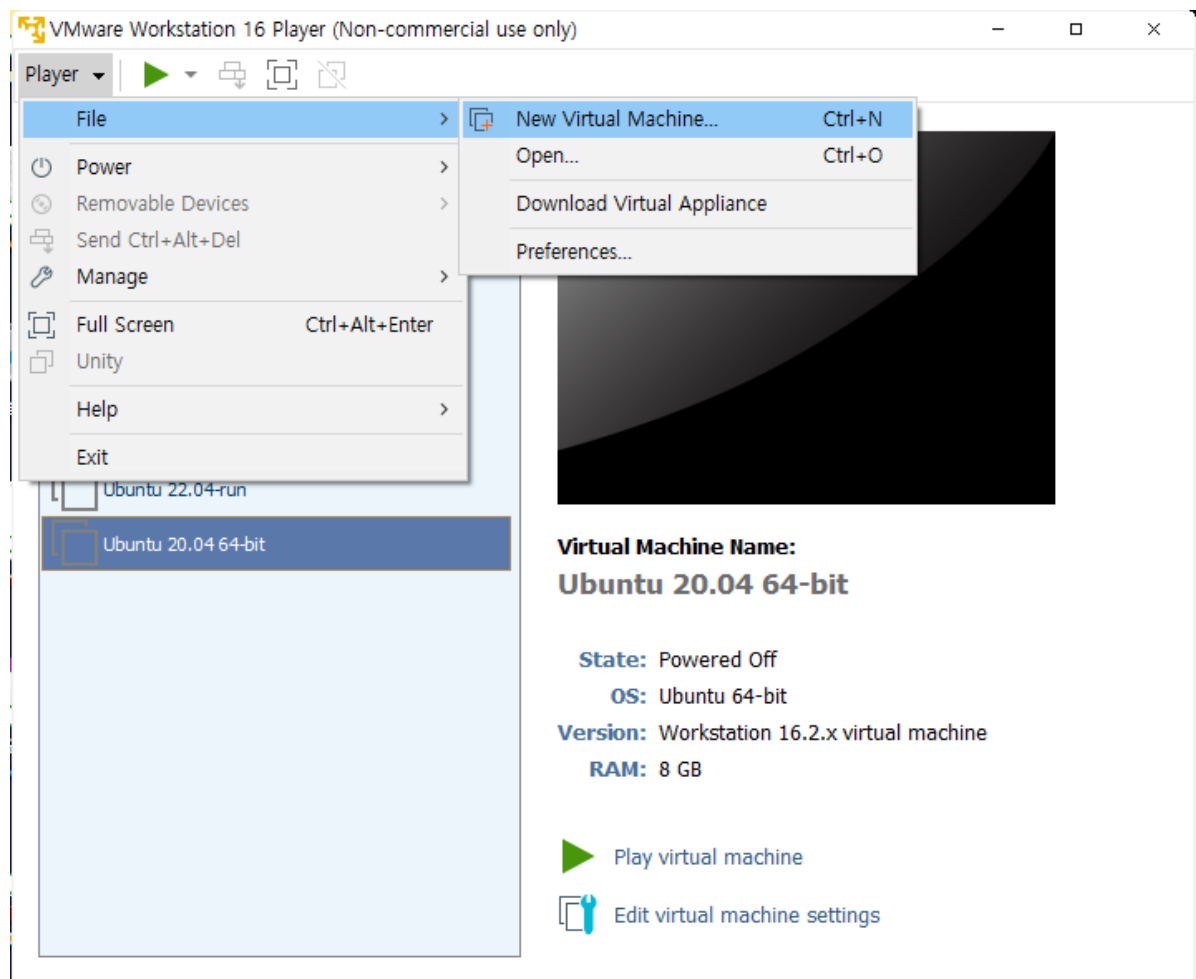
따라서 CentOS 를 설치 하려면 [볼트 Site](#)에서 (<https://vault.centos.org/>) ISO Image 를 Download 합니다.

실습에 사용할 CentOS 8.4 의 Download 주소는 https://vault.centos.org/8.4.2105/isos/x86_64/ 입니다.



ISO Image 의 Download 가 완료 되었으면 CentOS 설치를 시작 합니다.

4.1.1 VMware 를 시작 하고 Plaper -> File -> New Virtual Machine 을 선택 합니다.



4.1.2 ISO Image 파일 선택

다음으로 Installer disc image file(iso) 를 선택 후 Borwse... 버튼을 Click 하여 Download 해 놓은 ISO 파일을 선택 합니다.

New Virtual Machine Wizard

×

Welcome to the New Virtual Machine Wizard

A virtual machine is like a physical computer; it needs an operating system. How will you install the guest operating system?

Install from:

☐ Installer disc:

DVD RW 드라이브 (E:)

▼

☒ Installer disc image file (iso):

D:\Temp\CentOS-8.4.2105-x86_64-dvd1.iso

▼

Browse...

☐ I will install the operating system later.

The virtual machine will be created with a blank hard disk.

CentOS 8 64-bit detected.

This operating system will use Easy Install. [\(What's this?\)](#)

Help

< Back

Next >

Cancel

4,1.3 사용자 기본 정보를 입력 합니다.

New Virtual Machine Wizard

×

Easy Install Information

This is used to install CentOS 8 64-bit.

Personalize Linux

Full name:

CentOS-8.4

User name:

testuser

Password:

●●●●●●

Confirm:

●●●●●●

☐ This password is for both user and root accounts.

Help

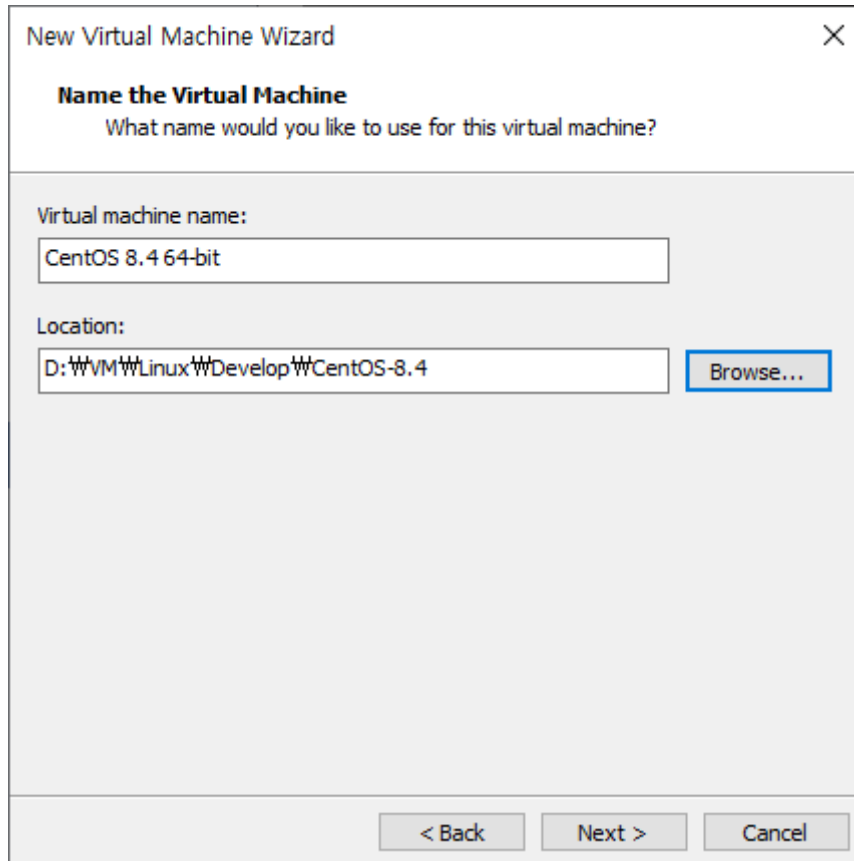
< Back

Next >

Cancel

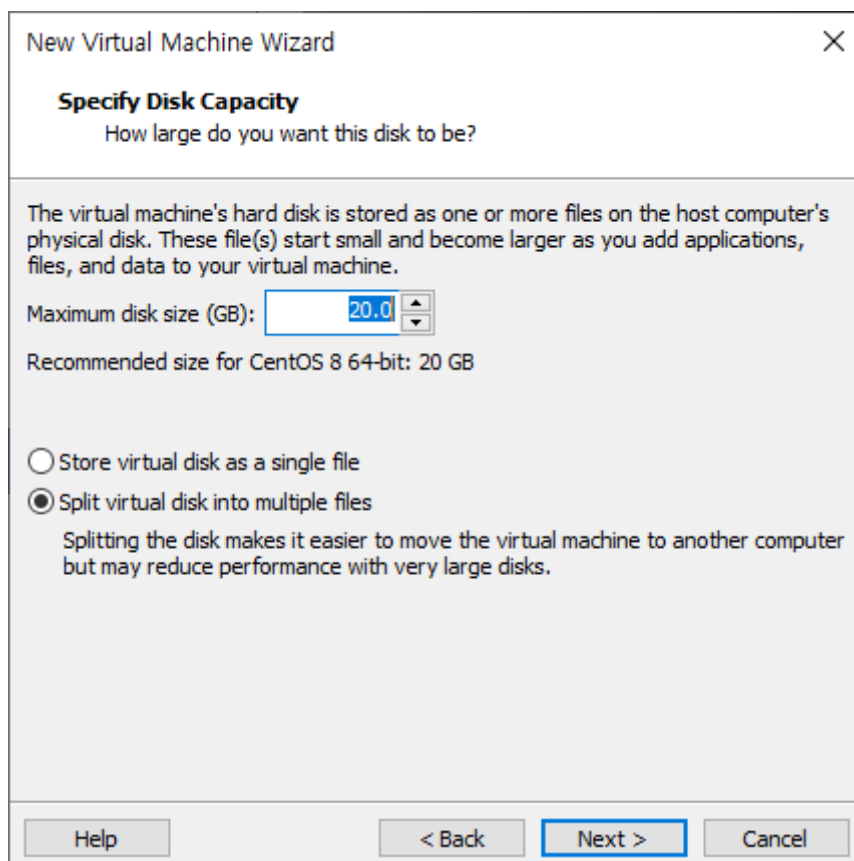
4.1.4 설치될 경로 선택

다음으로 가상머신 이름을 입력 하고 설치될 경로를 선택 합니다.



The screenshot shows the 'New Virtual Machine Wizard' window, specifically the 'Name the Virtual Machine' step. The window title is 'New Virtual Machine Wizard' with a close button (X) in the top right corner. Below the title bar, the section header is 'Name the Virtual Machine' followed by the question 'What name would you like to use for this virtual machine?'. There are two input fields: 'Virtual machine name:' with the text 'CentOS 8.4 64-bit' and 'Location:' with the text 'D:\VM\Linux\Develop\CentOS-8.4'. To the right of the 'Location' field is a 'Browse...' button. At the bottom of the window are three buttons: '< Back', 'Next >', and 'Cancel'.

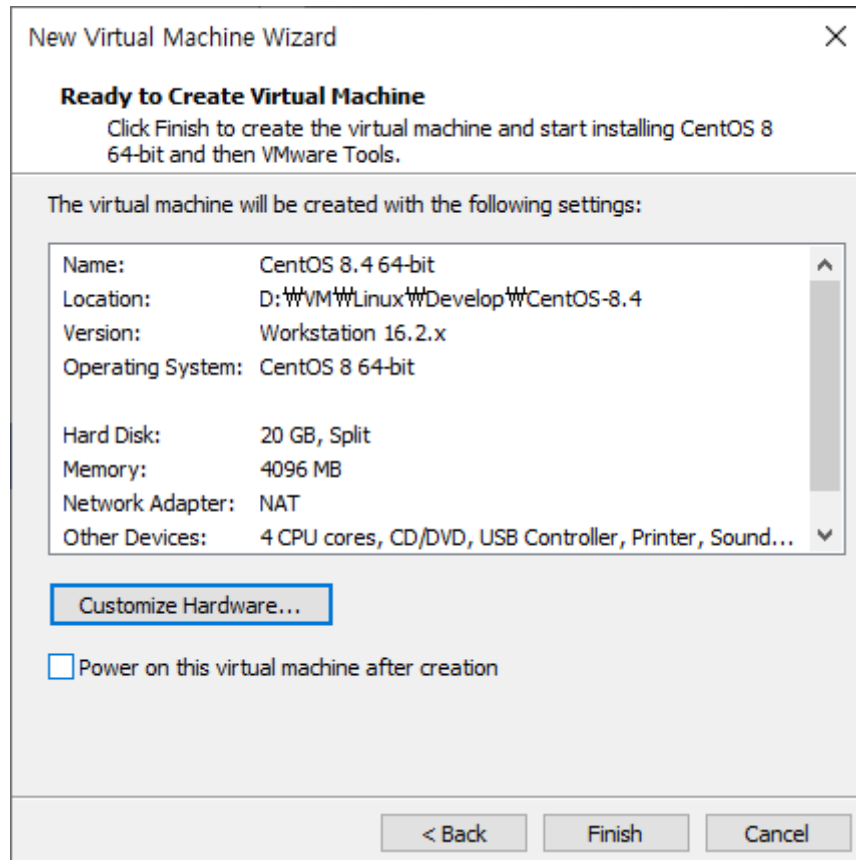
4.1.5 Linux 에서 사용할 Disk 용량을 선택 합니다.



The screenshot shows the 'New Virtual Machine Wizard' window, specifically the 'Specify Disk Capacity' step. The window title is 'New Virtual Machine Wizard' with a close button (X) in the top right corner. Below the title bar, the section header is 'Specify Disk Capacity' followed by the question 'How large do you want this disk to be?'. A paragraph of text explains: 'The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.' Below this text is a 'Maximum disk size (GB):' label followed by a numeric input field containing '20.0' and a spinner control. Below the input field is the text 'Recommended size for CentOS 8 64-bit: 20 GB'. There are two radio button options: 'Store virtual disk as a single file' (which is unselected) and 'Split virtual disk into multiple files' (which is selected). Below the selected option is a note: 'Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.' At the bottom of the window are four buttons: 'Help', '< Back', 'Next >', and 'Cancel'.

선택된 Disk 용량은 추후 필요에 의해 확장이 가능 하고, 처음 OS Image 생성시 선택된 용량 만큼 Size 를 차지하지 않으며, 사용중 용량이 늘어나 설정된 용량 까지 증가하게 됩니다.

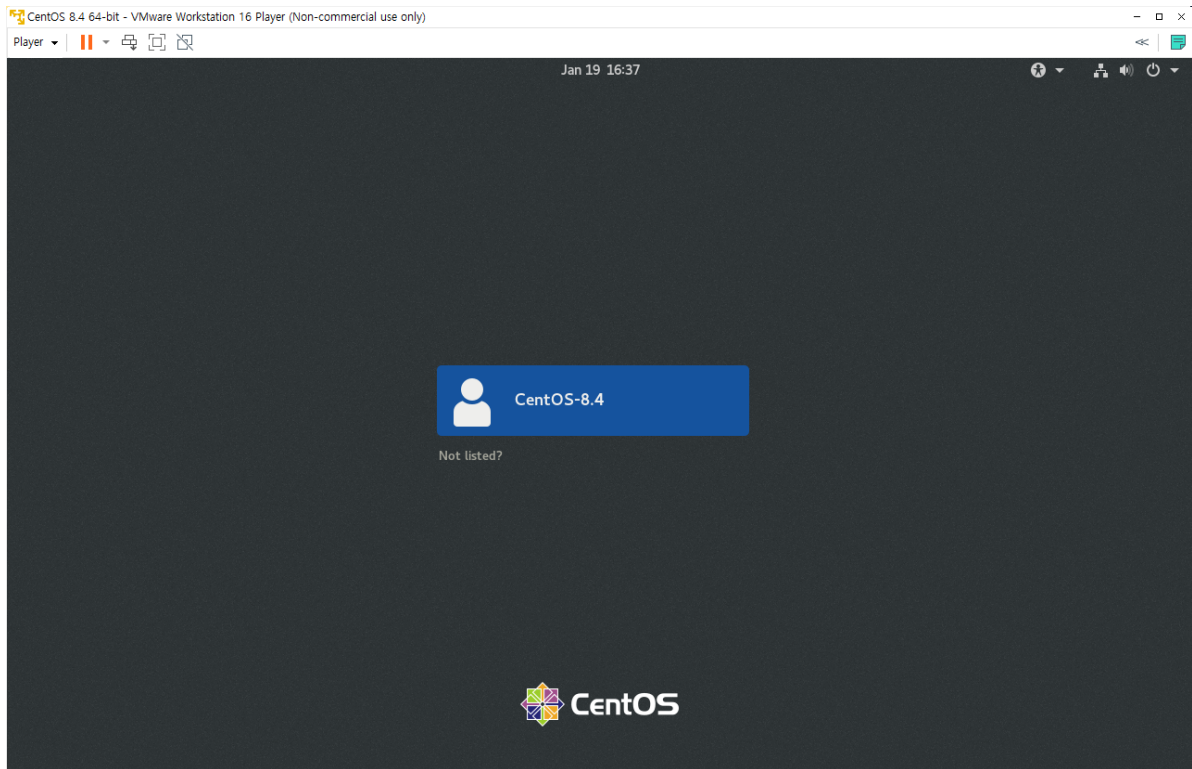
4.1.6 Customize Harware



필요시 Customize Harware 버튼을 Click 하여, CPU 수량, Memory Size, Disk Size 등을 추가로 설정 할 수 있습니다.

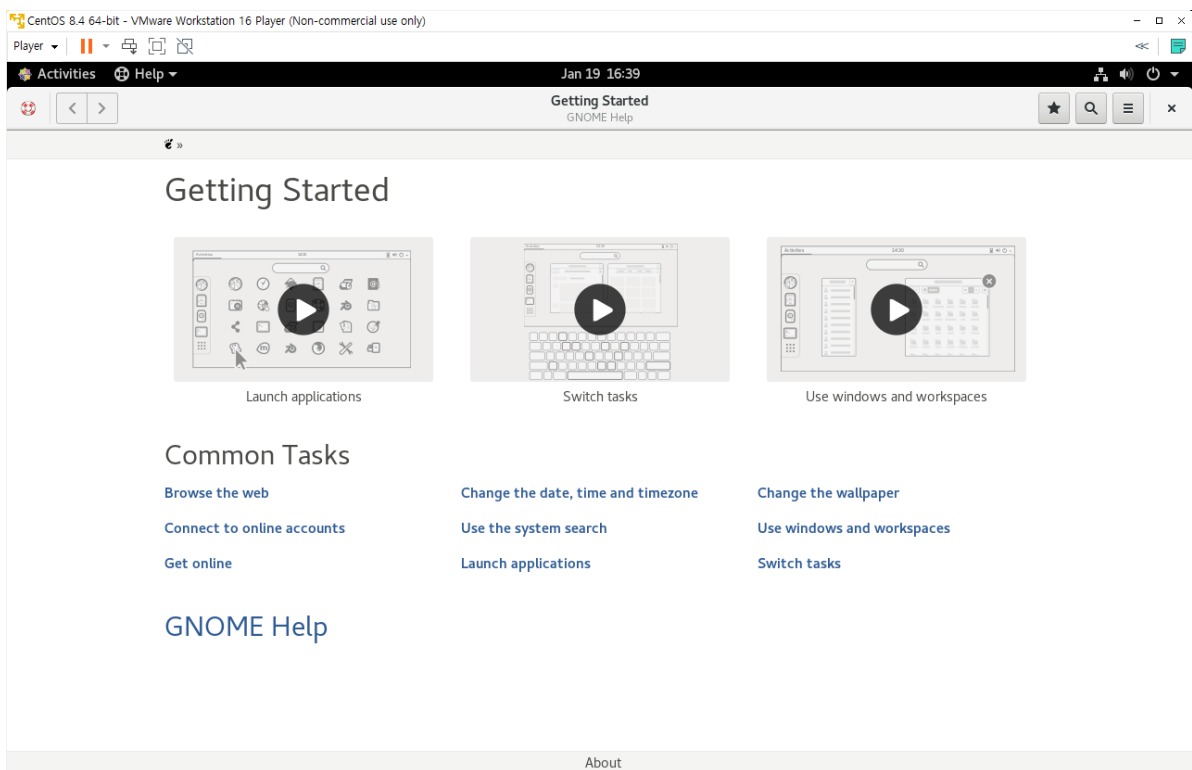
위 화면 에서 Finish Button 을 Click 하면 CentOS Linux 의 설치가 시작 됩니다.

4.1.7 설치 완료 후 재 시작

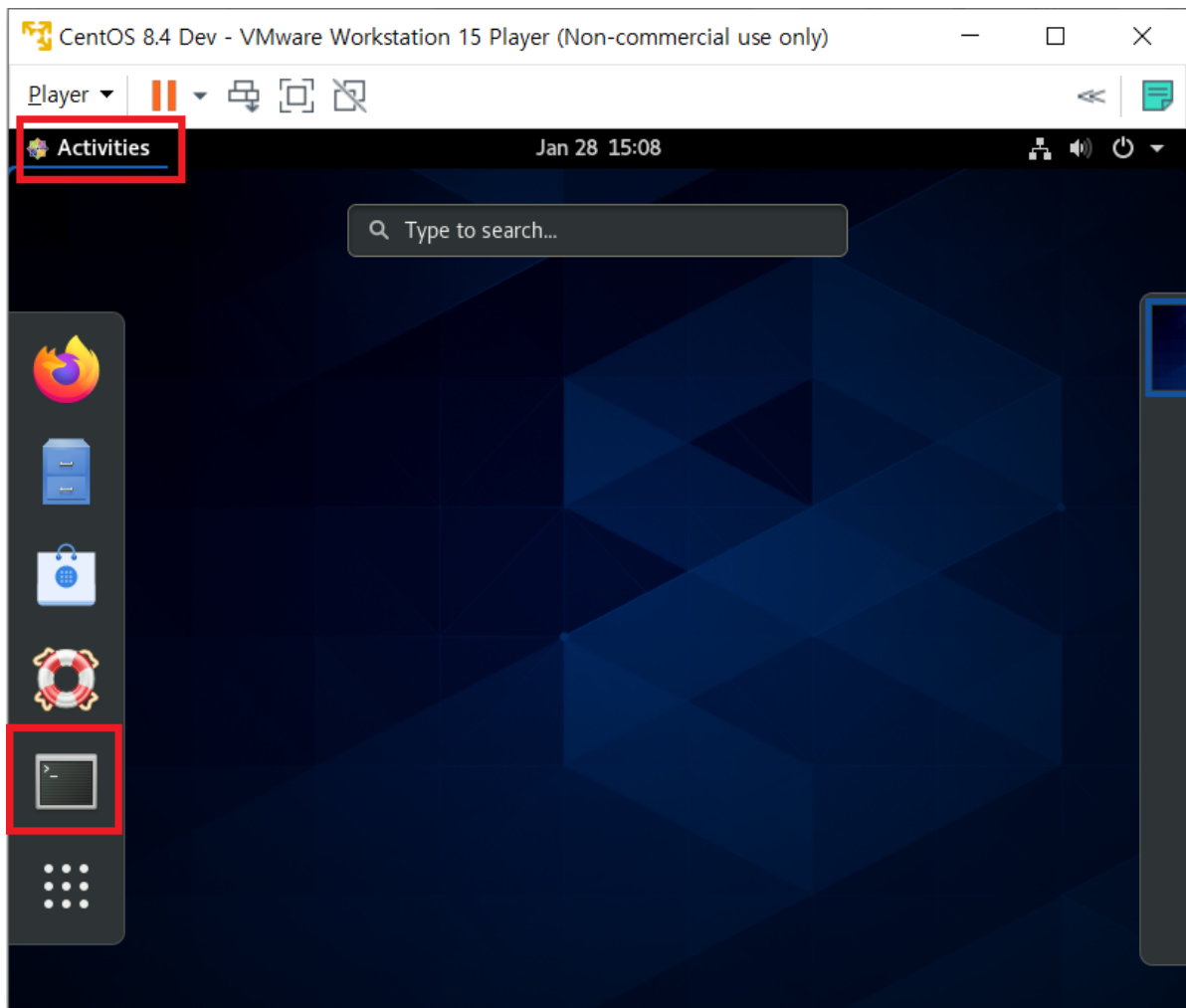


설치 완료 후 OS 재 시작 되면 위의 화면으로 전환 됩니다. 이때 사용자를 선택 후 설치과정 에서 설정 한 Password 를 입력 하여 Login 을 진행 하면 됩니다.

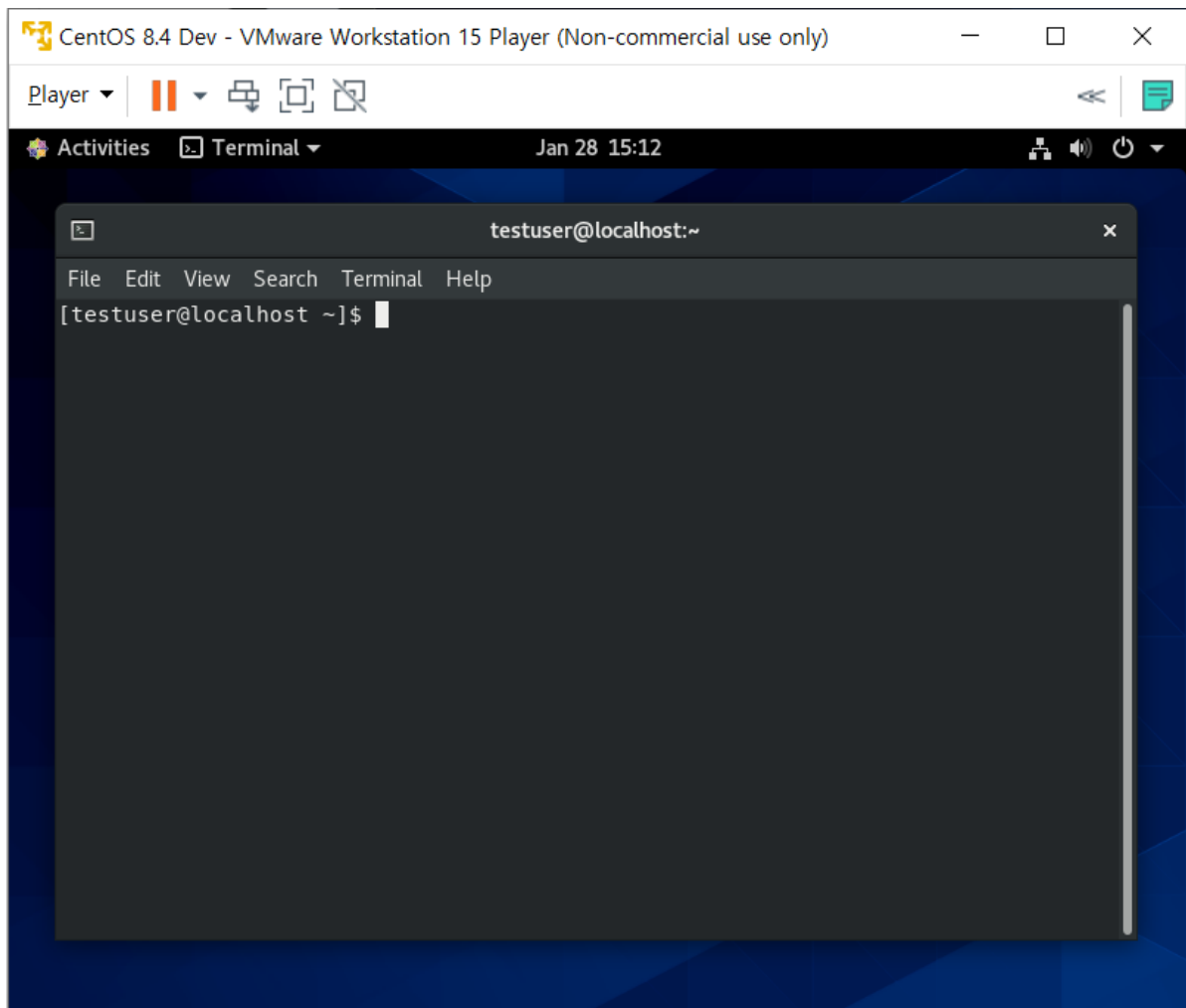
처음 CentOS 8.4 가 시작 되면 다음과 같은 화면이 나타나는데 이때 'x' 를 눌러 화면을 닫으면 됩니다.



다음으로 Main 화면이 나타나면 Activities -> Terminal 을 눌러 Linux Terminal 을 열 수 있습니다.



이제부터 대부분의 작업은 Terminal 창 에서 실행 됩니다.



5. Linux OS 주요 명령어 소개

Linux 환경에서 C/C++ 개발을 위한 대부분의 작업은 Terminal 창에서 이루어 집니다. 하지만 Linux 자체의 Terminal은 사용 하기가 불편 하므로 대부분의 개발자는 외부의 Windows 환경에 Terminal App을 설치하여 사용 하고 있습니다. Windows 환경에 설치한 Terminal App을 이용하여 Linux 에 Login 하기 위해서는 Telnet 또는 SSH Protocol 을 이용하여 접속을 하여야 합니다. 현재는 보안상의 이유로 Telnet 은 대부분 사용 하지 않고 SSH 를 사용 하고 있습니다.

Telnet Protocol 은 암호화 되지 않은 방식 으로 Windows Terminal App 과 통신을 하는 방식이고 SSH 는 암호화 하여 통신을 수행 하는 방식으로 해킹 으로 부터 좀더 안전합니다.

SSH 란? (What is SSH?)

시큐어 셸(Secure SHell, SSH)은 네트워크 상의 다른 컴퓨터에 로그인하거나 원격 시스템에서 명령을 실행하고 다른 시스템으로 파일을 복사할 수 있도록 해주는 응용 프로그램 또는 그 프로토콜을 가리킨다. 즉, 네트워크 프로토콜 중 하나로 컴퓨터와 컴퓨터가 인터넷과 같은 Public Network를 통해서 서로 통신을 할 때 보안적으로 안전하게 통신을 하기 위해 사용하는 프로토콜이다.

SSH 방식으로 Linux 에 접속 하기 위해서는 해당 Linux 에 SSH Service 가 기동되어 있어야 하고, 해당 Linux 의 IP Address 을 알아야 합니다.

5.1 IP Address 확인

Linux Terminal 에서 다음 명령(ifconfig)을 실행 하여 IP Address 를 확인 합니다.

```
[testuser@localhost ~]$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.211.130 netmask 255.255.255.0 broadcast 192.168.211.255
    inet6 fe80::20c:29ff:fe5e:d0 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:fe:5e:d0 txqueuelen 1000 (Ethernet)
    RX packets 1484 bytes 1741219 (1.6 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 885 bytes 67761 (66.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 4 bytes 240 (240.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 4 bytes 240 (240.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
    ether 52:54:00:fb:59:46 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[testuser@localhost ~]$
```

위 명령 실행 결과 해당 Linux 의 IP Address 는 ens33 network card 의 inet 192.168.211.130 즉 **192.168.211.130** 으로 확인이 되었습니다.

5.2 ssh Daemon 실행 확인

그리고 다음 명령을 실행 하여 SSH Server 가 실행되어 있는지 확인 합니다.

```
systemctl status sshd
```

결과 확인

```
[testuser@localhost ~]$ systemctl status sshd
• sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset:
   enabled)
   Active: active (running) since Sat 2023-01-28 15:00:15 PST; 45min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Main PID: 1207 (sshd)
    Tasks: 1 (limit: 23506)
   Memory: 2.0M
    CGroup: /system.slice/sshd.service
            └─1207 /usr/sbin/sshd -D -oCiphers=aes256-gcm@openssh.com,chacha20-
poly1305@openssh.com,aes256-ctr,aes256-cbc,aes128-gcm@openssh.com,aes128-
ctr,aes128-cbc -oMACs=hm>
[testuser@localhost ~]$
```

CentOS 8.4 에는 기본적으로 ssh가 설치 및 활성화 되어 있습니다.

혹시 ssh 가 설치되어 있지 않은 경우 별첨 [CentOS 에 SSH 설치하기](#) 문서를 참조하여 설치 하시기 바랍니다.

5.2.2.1 CentOS 7 이후의 방화벽 변경 사항

CentOS 7 부터는 firewalld 서비스가 기본 방화벽으로 활성화 되도록 적용 되었습니다.

이전 Version 에는 iptables를 사용했는데 CentOS 7 부터 firewalld 라는 방화벽 시스템이 기본으로 바뀜

5.2.2 방화벽 열기

```
# 80 포트 열기
> firewall-cmd --add-port=80/tcp

# 80,81 포트 열기
firewall-cmd --add-port={80,81}/tcp

# 80 ~ 90 포트 열기
firewall-cmd --add-port=80-90/tcp

# 80 포트 닫기
firewall-cmd --remove-port=80/tcp

# 포트 변경사항 적용하기
firewall-cmd --runtime-to-permanent

# 포트 열려있는 사항 확인하기
firewall-cmd --list-port

# 포트 변경사항 적용 후 방화벽 reload
firewall-cmd --reload
```

```
# SSH 포트를 열기 위해서는 다음 명령을 실행 한다.
firewall-cmd --permanent --zone=public --add-port=22/tcp
firewall-cmd --reload
firewall-cmd --list-port

# 시작
systemctl start firewalld

# 중지
systemctl stop firewalld

# 재시작
firewall-cmd --reload
```

5.3 Windows SSH Terminal 를 이용하여 Linux 에 접속 하기

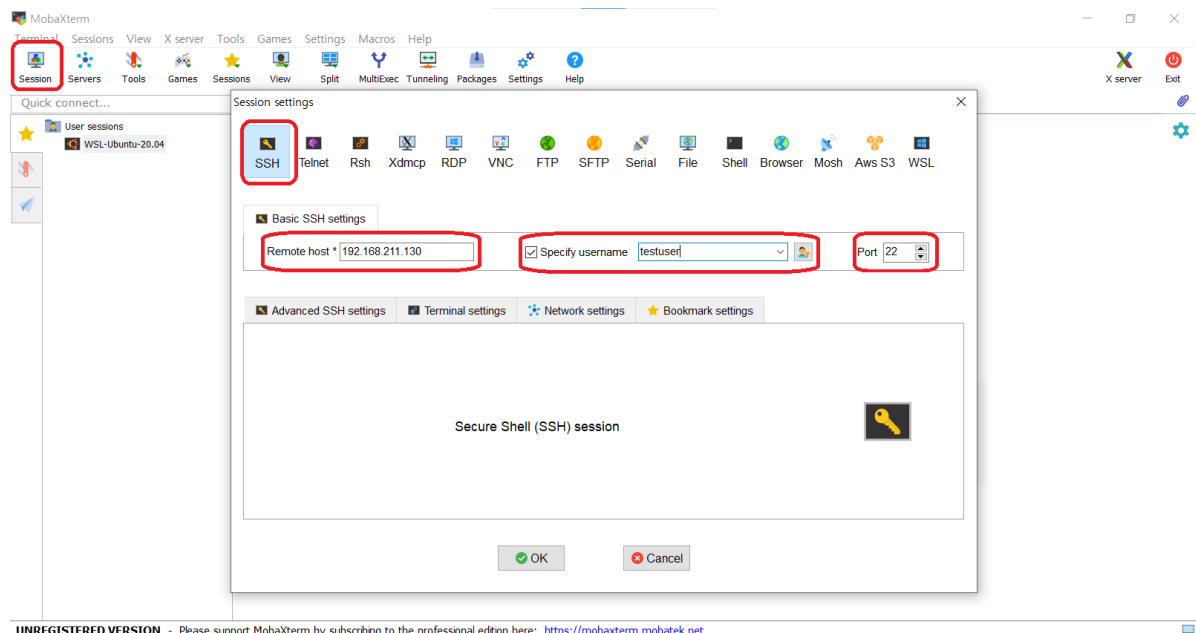
이번에는 Windows SSH Terminal 중의 하나인 mobaxterm 을 이용하여 설치된 CentOS Linux 에 접속 하는 방법을 기술 합니다.

mobaxterm 은 Linux 와 접속 하여 Console 환경의 Terminal 기능을 수행 하고 X-Terminal 기능도 수행 할 수 있는 Client App 입니다.

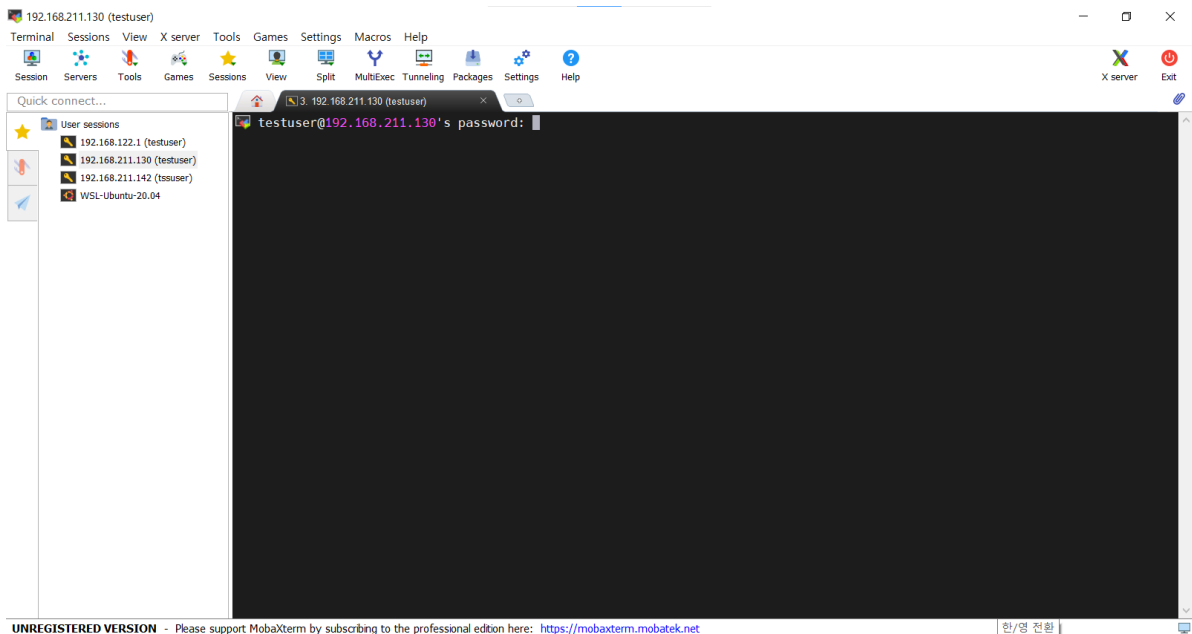
Home Page: <https://mobaxterm.mobatek.net>

mobaxterm 의 설치 과정은 일반 software 의 설치와 유사한 방법으로 설치 할 수 있으므로 생략 하고 설치되어 있는 mobaxterm 을 이용하여 위에서 파악된 CentOS-8.6 의 IP 주소인 192.168.211.130 에 접속 하는 방법을 설명 합니다.

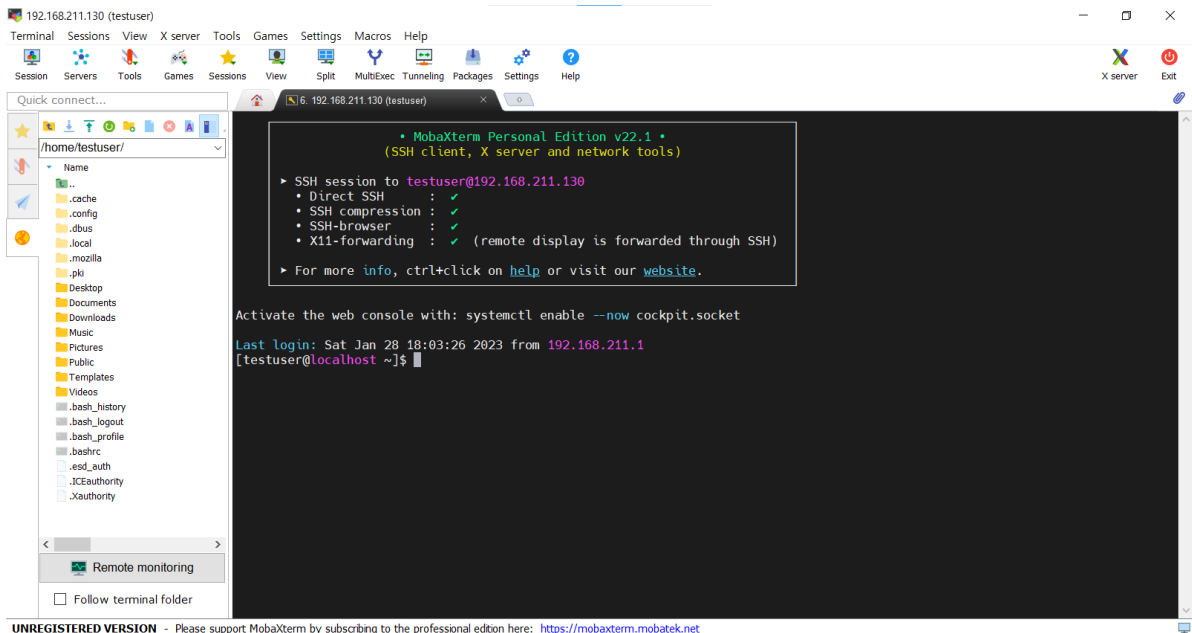
mobaxterm 을 실행 한 후 Session 버튼을 눌러 접속할 SSH 주소를 추가 합니다.



설정된 SSH Connection 정보를 이용해 SSH 접속을 시도 하면 Password 를 묻는 창이 나타납니다.



이때 설치과정에서 지정 하였던 password 를 입력 하고 key 를 누르면 접속이 완료됩니다.



이제부터 대부분은 명령은 이 창 에서 실행 됩니다.

5.3 directory 관련 명령어 소개

컴퓨터에서 폴더(Folder)와 디렉토리(Directory)는 거의 똑같은 말입니다. 99% 정도의 동의어입니다. 폴더는 원래, 여러 장의 서류들을 합쳐서 보관/휴대하기 위한 '접을 수 있는 겉표지'입니다. 디렉토리는 주소록이나 전화번호부 같은 '목록'을 지칭합니다.

도스(MS-DOS)나 리눅스/유닉스의 텍스트모드 프롬프트에서는 디렉토리라고 부르고, 윈도우나 매킨토시 같은 그래픽 환경에서는 폴더라고 부르는 것이 보통입니다.

OS 별 경로 구분자

windows 의 경우 경로 구분자를 역 슬래쉬(\\) 를 사용하고 Linux 및 Unix 에서는 슬래쉬(/) 를 사용합니다.

5.3.1 directory 만들기 명령

mkdir의 뜻은 make directory의 약자로 디렉토리(폴더)를 생성할 때 사용하는 명령어입니다. 실제로 사용할 때는 **mkdir** 대신 **md** 를 많이 사용 합니다. linux terminal 에서 사용하는 명령어중 다수는 약어를 사용하고 있습니다.

사용 방법:

mkdir [옵션] [생성 할 디렉토리]

사용 예: new_folder 디렉토리 생성

```
mkdir new_folder
```

경로를 지정하지 않으면 현 위치를 Default로 합니다.

생성할 디렉토리를 여러개 나열하여 한번의 명령어로 복수의 디렉토리를 생성할 수 있습니다.

자주 사용하는 옵션

- **-m** : 디렉토리를 생성할 때 권한을 설정합니다. (default : 755)
- **-p** : 상위 경로도 함께 생성합니다.
- **-v** : 디렉토리를 생성하고 생성된 디렉토리에 대한 메시지를 출력합니다.

디렉토리안에 디렉토리를 만들고 싶을때

```
mkdir folder/sub_folder
```

이때 folder directory 가 없으면 오류가 발생 합니다.

오류를 해결 기 위한 방법은 다음과 같습니다.

```
mkdir folder
mkdir folder/sub_folder
```

또는

```
mkdir -p folder/sub_folder
```


5.3.2 디렉토리 이동

cd 의 뜻은 change directory의 줄임말로 **현재 작업하고 있는 디렉토리의 위치를 이동**하는 명령어 입니다. cd 명령어 뒤에 디렉토리 이동경로를 입력해주시면 해당 디렉토리로 즉시 이동합니다. 이 디렉토리 이동경로를 입력하실때는 키보드 TAB 버튼을 함께 사용하시면 어느정도 경로를 자동으로 지정해주기에 좀 더 편하게 디렉토리 이동을 하실 수 있습니다.

사용 방법:

cd [이동 할 디렉토리 경로]

사용 예: new_folder 로 이동

```
cd new_folder
```

절대 경로와 상대 경로

경로의 형식에는 **절대 경로와 상대경로**가 있습니다. 절대 경로란 FULL NAME을 시작(ROOT)부터 끝까지 다 입력한 것을 말합니다. 예를 들자면 /home/user/develop/jtaewu 이렇게 말이죠.

상대 경로는 현재 위치한 폴더를 기준으로 이동하고자 하는 디렉토리로 바로 이동할 때 사용하는 방법입니다. 상대 경로의 개념을 비유적으로 쉽게 이해하는 방법은 윈도우 폴더의 뒤로 가기/앞으로 가기 기능을 떠올리면 됩니다. CD의 디렉토리 경로 인자 값에 특수기호를 넣어 다양한 액션을 취할 수 있습니다.

- **cd ..**: 상위 디렉토리로 이동합니다. 예를 들어 현재 위치가 /user/jtaewu라면 /user로 이동합니다.
- **cd .**: 현재 위치한 폴더로 이동합니다. 사실상 기능은 새로고침과 동일합니다.
- **cd -**: 이전에 위치했던 폴더로 이동합니다. 윈도우의 뒤로 가기와 동일합니다.
- **cd /**: ROOT 디렉토리로 이동합니다.
- **cd ~**: 홈 디렉토리로 이동합니다.

5.3.3 디렉토리 삭제

Linux 에서 directory 를 지울때 사용하는 명령은 rmdir 입니다.

rmdir 은 remove empty directories 의 약자로 비어있는 directory 를 지우는 명령 입니다.

사용 방법:

rmdir [삭제 할 디렉토리 경로]

사용 예: new_folder 삭제

```
rmdir new_folder
```

비어있지 않은 directory 를 지우기

위의 설명 에서 rmdir 은 비어있는 directory 를 지우라는 명령으로 설명을 하였는데, 비어 있지 않은 directory 를 강제로 지우는 명령은 없을까요? 다음 명령으로 지울 수 있습니다.

```
rm -rf new_folder
```

위 명령은 **new_folder** 를 포함하여 하위 경로의 모든 파일 및 **directory** 를 지우는 명령으로 아주 위험 할 수 있습니다.

5.4 Linux 에서 사용자 생성 및 삭제

5.4.1 linux 에서 root user 와 일반 user 차이

Linux의 'root' 사용자는 관리 권한이 있는 슈퍼유저이며 일반 사용자는 시스템에 대한 액세스가 더 제한적입니다. root 사용자는 모든 파일과 시스템 설정에 액세스하고 수정할 수 있는 반면 일반 사용자는 액세스가 제한되어 자신의 홈 디렉토리 내에서만 파일과 설정을 수정할 수 있습니다. 또한 root 사용자만 소프트웨어를 설치 및 업데이트하고 네트워크 구성 또는 새 사용자 추가와 같은 시스템 전체 작업을 수행할 수 있습니다. 일반적으로 일상적인 작업에는 일반 사용자 계정을 사용하고 관리 작업을 수행하는 데 필요한 경우에만 루트 사용자로 전환하는 것이 가장 좋은 방법으로 간주됩니다.

NOTE

요즘의 대부분의 Linux 에서는 sudo 명령을 이용하여 superuser 권한이 있어야 실행 할 수 있는 명령을 실행 할 수 있습니다. 대표적으로 Ubuntu 에서 사용 하며, 현재는 CentOS 에서도 사용 할 수 있습니다. Ubuntu 에서는 설치시 등록된 default user 가 sudo 권한을 가지고 있지만 CentOS 에서는 별도의 설정이 필요 합니다.

CentOS 에서 사용자 추가 및 삭제는 superuser 권한을 가지고 있는 root user 가 작업을 하는것이 일반적 입니다.

일반 user 에서 root user 로 switching 하는 방법은 su 명령을 이용합니다.

```
su - root
```

위 명령을 실행하면 root user 로 switch 됩니다. 이때 root user 의 password 를 묻게 되는데 처음 CentOS 를 설치한 경우는 default user 의 password 와 동일하게 설정 되어 있습니다.

TIP

su 명령 다음에 나오는 - 는 다른 사용자의 계정으로 완전히 전환하고, 전환한 사용자의 환경설정을 불러옵니다. - 옵션이 없는 경우는 현재 사용자의 환경(PATH, 등등)을 사용하고 권한만 전환된 사용자의 권한을 이용합니다.

초보 리눅서는 - 옵션 사용 여부에 따라 App 이 상이하게 동작 할 때 당황 할 수 있으니 주의 하시기 바랍니다.

5.4.2 Linux 에서 사용자 생성

Linux 에서 사용자를 생성 하는 명령이 2가지 있습니다.

useradd 명령어와 adduser 명령어 입니다. 이 두가지 명령의 차이는 useradd 는 계정을 생성할 때 필요한 모든 설정들을 명시해줘야 하고, adduser 는 필요한 대부분의 설정을(user 의 home directory 등) default 로 사용 합니다. 따라서 사용하기 쉬운 adduser 를 사용하길 권장 합니다.

다음 명령을 이용해 사용자를 생성 합니다.

```
[root@localhost ~]# adduser user1
```

위에서 `[root@localhost ~]#` 까지 는 **prompt** 입니다. 따라서 `#`은 입력하는것이 아닙니다.

위 명령을 실행 하면 user1 이라는 user 가 생성되고 user1 의 home directory 는 '/home/user1' 이 됩니다.

이때 password 는 생성되지 않았으므로 root user 가 su 명령을 사용하여 switch 하는 방법 이외에는 user1 으로 login 하는 방법이 없습니다.

root user 권한으로 다음 명령을 실행 하여 user1 의 password 를 설정 합니다.

```
[root@localhost ~]# passwd user1
Changing password for user user1.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]#
```

password 는 8 자리 이상을 사용하는것을 권장 하고 있습니다.

5.4.4 Linux 에서 사용자 삭제

linux 에서 사용자는 다음 명령을 삭제 합니다.

명령어:

userdel [옵션] [사용자명]

사용 예:

```
[root@localhost ~]#
[root@localhost ~]# userdel -r user1
[root@localhost ~]#
```

위의 경우 '-r' 옵션을 지정 하지 않으면 사용자의 home directory 가 삭제되지 않고 남아 있습니다.

즉 '/home/user1' directory 가 삭제되지 않고 남아 있습니다.

5.5 Linux 의 퍼미션(permission)

Linux에서 권한은 파일 및 디렉터리에 대한 액세스를 제어하는 방법입니다. 권한에는 읽기, 쓰기 및 실행의 세 가지 유형이 있습니다. 각 파일 및 디렉토리에는 **소유자**, **그룹** 및 **기타 사용자**에게 할당된 권한이 있습니다. 소유자는 파일이나 디렉토리를 만든 사용자이고, 그룹은 사용자 모음이며 나머지는 시스템의 다른 모든 사용자입니다.

권한은 -rwxrwxrwx와 같은 일련의 문자로 표시될 수 있습니다. 여기서 각 문자는 소유자, 그룹 및 기타에 대한 권한을 나타냅니다. 예를 들어 문자 'r'은 읽기(4) 권한을, 'w'는 쓰기(2) 권한을, 'x'는 실행(1) 권한을 나타냅니다. 글자가 없으면 권한이 부여되지 않았음을 나타냅니다.

chmod 명령을 사용하여 파일 또는 디렉토리에 대한 권한을 변경할 수 있습니다. chown 명령을 사용하여 파일 또는 디렉토리의 소유자 및 그룹을 변경할 수도 있습니다.

시스템을 보호하고 중요한 정보에 대한 무단 액세스를 방지하는 데 도움이 되므로 권한을 올바르게 이해하고 설정하는 것이 중요합니다.

5.5.1 chmod (권한 변경)

Linux에서 chmod 명령은 파일 또는 디렉토리에 대한 권한을 변경하는 데 사용됩니다. 파일 권한은 누가 파일을 읽거나 쓰거나 실행할 수 있는지를 결정합니다.

chmod 명령은 다음 구문을 사용하면 됩니다.

```
chmod [permissions] [file or directory]
```

예를 들어 myfile.txt라는 파일에 소유자에게 읽기 및 쓰기 권한을 부여하고 다른 모든 사람에게 읽기 전용 권한을 부여하려면 다음 명령을 사용합니다.

```
chmod 644 myfile.txt
```

권한은 3자리 숫자로 표시되며 각 숫자는 각각 소유자, 그룹 및 기타 사용자의 권한에 해당합니다. 숫자는 다양한 값의 합으로 구성됩니다.

- 7 = 읽기, 쓰기 및 실행 권한 (rwx)
- 6 = 읽기 및 쓰기 권한 (rw-)
- 5 = 읽기 및 실행 권한 (r-x)
- 4 = 읽기 및 실행 권한 (r--)
- 2 = 쓰기 전용 권한 (-w-)
- 1 = 실행 전용 권한 (--x)

chmod를 사용하는 또 다른 방법이 있습니다. 문자 u,g,o,a,+,-,=를 사용하여 각각 사용자, 그룹, 기타, 모두, 추가, 제거 및 설정을 나타냅니다.

예를 들어,

```
chmod u+x myfile.txt
```

이 명령은 소유자에게 myfile.txt 파일에 대한 실행 권한을 부여합니다.

루트 사용자는 모든 파일에 대한 권한을 변경할 수 있는 반면 일반 사용자는 자신이 소유한 파일에 대한 권한만 변경할 수 있다는 점에 유의해야 합니다.

5.5.2 chown (소유권 변경)

Linux에서 chown 명령은 **파일 또는 디렉토리의 소유권을 변경하는 데 사용됩니다**. Linux의 모든 파일과 디렉토리는 특정 사용자와 그룹이 소유합니다. chown 명령을 사용하여 파일 또는 디렉토리의 사용자 및 그룹 소유권을 모두 변경할 수 있습니다.

chown 명령의 기본 구문은 다음과 같습니다.

```
chown [options] [user][:group] [file or directory]
```

예를 들어, myfile.txt라는 파일의 소유권을 사용자 john과 그룹 users로 변경하려면 다음 명령을 사용합니다.

```
chown john:users myfile.txt
```

다음 명령을 사용하여 사용자 또는 그룹 소유권만 변경할 수도 있습니다.

```
chown john myfile.txt
```

이 명령은 사용자 소유권만 john으로 변경하고 그룹 소유권은 변경하지 않은 상태로 둡니다.

chown 명령에는 디렉토리 내의 모든 파일 및 디렉토리의 소유권을 재귀적으로 변경하거나 가리키는 파일이 아닌 심볼릭 링크의 소유권을 변경하는 것과 같은 고급 소유권 변경을 수행할 수 있는 몇 가지 옵션도 있습니다.

루트 사용자 또는 파일 소유자만 파일 소유권을 변경할 수 있고 루트 사용자만 디렉터리 소유권을 변경할 수 있다는 점에 유의해야 합니다.

5.5.3 chgrp (그룹 소유권 변경)

Linux에서 chgrp 명령은 파일 또는 디렉토리의 그룹 소유권을 변경하는 데 사용됩니다. Linux의 모든 파일과 디렉토리는 특정 사용자와 그룹이 소유합니다. chgrp 명령은 사용자 소유권을 변경하지 않고 파일 또는 디렉토리의 그룹 소유권을 변경하는 데 사용됩니다.

chgrp 명령의 기본 구문은 다음과 같습니다.

```
chgrp [options] [group] [file or directory]
```

예를 들어 myfile.txt라는 파일의 그룹 소유권을 그룹 **'users'** 로 변경하려면 다음 명령을 사용합니다.

```
chgrp users myfile.txt
```

또한 chgrp 명령에는 디렉토리 내의 모든 파일 및 디렉토리의 그룹 소유권을 재귀적으로 변경하거나 심볼릭 링크가 가리키는 파일이 아닌 심볼릭 링크의 그룹 소유권을 변경하는 것과 같은 고급 그룹 소유권 변경을 수행할 수 있는 몇 가지 옵션이 있습니다.

루트 사용자 또는 파일 소유자만 파일의 그룹 소유권을 변경할 수 있고 루트 사용자만 디렉터리의 그룹 소유권을 변경할 수 있다는 점에 유의해야 합니다. 또한 사용자는 파일/디렉토리에 할당되는 그룹의 구성원이어야 합니다.

6. 환경변수

Linux에서 '환경'은 운영 체제 및 운영 체제에서 실행되는 프로그램의 동작에 영향을 미치는 변수 및 설정 모음을 나타냅니다. 환경에는 실행 파일의 경로, 구성 파일의 위치 및 현재 사용자의 홈 디렉터리와 같은 다양한 변수가 포함될 수 있습니다.

Linux에서 가장 중요한 환경 변수 중 하나는 운영 체제가 실행 파일을 검색할 디렉터리를 결정하는 PATH 변수입니다. PATH 변수는 콜론(:)으로 구분된 디렉토리 목록입니다.

또 다른 중요한 환경 변수는 현재 사용자의 홈 디렉터리로 설정되는 HOME 변수입니다. 이 변수는 프로그램에서 구성 파일 또는 기타 데이터를 저장할 위치를 결정하는 데 자주 사용됩니다.

SHELL 변수는 사용자의 기본 셸을 지정합니다(예: /bin/bash).

터미널에서 env 또는 printenv 명령을 실행하여 현재 환경 변수를 볼 수 있습니다. 변수의 특정 값을 보려면 echo \$VAR_NAME을 사용할 수 있습니다.

터미널에서 내보내기 명령을 사용하여 환경 변수를 설정하거나 수정할 수도 있습니다. 예를 들어 MYVAR 변수를 myvalue 값으로 설정하려면 다음 명령을 사용합니다.

```
export MYVAR=myvalue
```

이 변경 사항은 현재 터미널 세션에만 적용됩니다. **변경 사항을 영구적으로 적용하려면 .bashrc, .bash_profile** 등과 같은 셸 구성 파일에 내보내기 명령을 추가해야 합니다.

LD_LIBRARY_PATH 환경변수는 '로더(Loader)'가 공유 라이브러리나 동적 라이브러리를 찾아야 할 때 어떤 경로를 찾아가야 하는지를 지정하는 환경변수다. 실행 파일을 찾아가는 PATH 환경변수의 라이브러리 버전이라고 생각하면 된다.

```
export LD_LIBRARY_PATH=/home/user/lib:/home/user/test/lib
```

6.1 환경변수 load

.bashrc 또는 .bash_profile 의 환경변수 를 수정 하였으면 다음번 login 이후에 적용이 된다, 하지만 재 login 않고 현재 접속된 환경 에서 설정을 적용 하려면 다음 명령을 실행 하면 됩니다.

```
source ~/.bashrc
source ~/.bash_profile
```

또는

```
. ~/.bashrc
. ~/.bash_profile
```

6.2 Linux 유용한 환경변수

linux bash shell 은 .bashrc 또는 .bash_profile 파일을 읽어서 환경 변수를 설정 합니다. 다음은 유용한 환경 변수 입니다.

6.2.1 alias(별칭)

Linux의 alias는 명령 또는 명령 집합에 대한 바로 가기입니다. alias를 생성하여 시간을 절약하고 자주 사용하는 명령을 단순화할 수 있습니다. 예: 'alias ls='ls --color=auto'는 'ls --color=auto' 명령에 대한 별칭 'ls'를 생성합니다. 별칭을 사용하려면 터미널에 'ls'를 입력하면 'ls --color=auto'가 실행됩니다. 별칭을 제거하려면 'unalias' 명령 다음에 alias 이름을 사용하십시오.

alias 는 명령어의 별칭 으로 사용 법은 다음과 같습니다.

.bashrc 또는 .bash_profile 에 다음 명령을 추가 합니다.

```
alias [alias 이름]='명령어'

# 예
alias src='cd ~/workspace/test_source/source/project/testproj'
```

위처럼 alias 를 지정 하고 적용(재 login 시 적용됨)이 되었으면 Terminal 창에서 src 입력 후 enter key 를 누르면 ~/workspace/test_source/source/project/testproj 경로로 이동 됩니다.

6.2.2 bash 함수

Linux의 bash 함수는 단일 이름으로 그룹화된 명령 집합입니다. 함수는 별칭과 비슷하지만 인수를 허용하고 더 복잡한 논리 및 제어 구조를 포함할 수 있습니다. 다음 구문을 사용하여 bash 셸에서 정의됩니다.

```
function function_name {
    commands
}
```

예를 들어, 다음 함수 'hello'는 사용자에게 인사를 출력합니다.

```
function hello {
    echo "Hello, $1"
}
```

다음과 같이 함수 이름 뒤에 임의의 인수를 입력하여 함수를 호출할 수 있습니다. 함수를 제거하려면 'unset' 명령 다음에 함수 이름을 사용하십시오.

```
> hello world
```

다음은 특정한 문자열을 포함하고 있는 파일을 찾아서 표시하는 bash 함수의 사용 예 입니다.

```
pfind()
{
    find . -name $1 | xargs egrep $2
}
```

사용법은 다음과 같습니다.

```
pfind *.cpp printf
```

위 명령은 현재 directory 아래(sub directory 포함) 에서 확장자가 .cpp 인 파일 중 printf 문자열을 포함하고 있는 줄을 파일명과 함께 보여 줍니다

6.2.3 PATH 환경 변수

Linux 에서 명령을 실행 하면 명령어가 OS 의 기본 directory 에 없는 경우는 환경 변수 PATH 에 기술된 순서로 명령어를 찾습니다. 현재 경로에 'testmain' 이라는 App 이 존재 하는 경우에도 testmain 이라고 입력 하면 testmain Not Found 라는 오류를 보여줍니다.

이때 오류를 해결 하기 위해 .bashrc 또는 .bash_profile 에 다음 명령을 추가 합니다.

```
export PATH=.:$PATH
```

설명:

- `.` : 현재 directory
- `:` : 여러개의 경로를 등록 할 경우 구분자
- `$PATH` : 기존에 등록되어 있던 환경변수 PATH 의 값

이제 적용이 완료되면 현재 경로의 파일은 `./` 를 입력 하지 않아도 참조 할 수 있습니다.

7. Linux 기타 명령어 및 팁

7.1 파일 압축 및 해제 (tar, gzip)

7.1.1 tar 명령을 이용한 아카이브 파일 생성

Linux의 tar 명령은 .tar 형식의 아카이브 파일을 생성, 추출 및 조작하는 데 사용됩니다. 다음은 tar 명령을 사용하는 방법에 대한 몇 가지 기본 예입니다.

- 디렉토리의 아카이브를 생성하려면 다음 구문을 사용하십시오.

```
tar -cvf archive.tar /path/to/directory
```

- 아카이브에서 파일을 추출하려면 다음 구문을 사용하십시오.

```
tar -xvf archive.tar
```

- 아카이브에서 특정 디렉토리로 파일을 추출하려면 다음 구문을 사용하십시오.

```
tar -xvf archive.tar -C /path/to/directory
```

- 기존 아카이브에 새 파일을 추가하려면 다음 구문을 사용하십시오.

```
tar -rvf archive.tar /path/to/new/file
```

- 아카이브의 내용을 나열하려면 다음 구문을 사용하십시오.


```
tar -tvf archive.tar
```

- 위의 예는 tar 명령으로 사용할 수 있는 많은 옵션의 몇 가지 예일 뿐입니다. 터미널에서 `man tar`를 실행하여 tar 매뉴얼 페이지에서 자세한 정보와 옵션을 찾을 수 있습니다.

tar utility 는 아카이브 파일만 생성 하며, 파일 size 가 줄어드는 압축을 하지 않습니다. 따라서 Linux 에서 파일을 다른곳 으로 복사 할 경우 size 를 줄이기 위해 gzip utility 를 사용하여 size 를 줄인 후 복사를 하는 방법을 많이 사용 합니다.

7.1.2 gzip 을 이용한 파일 압축

Linux의 gzip 명령은 파일을 압축하고 압축 해제하는 데 사용됩니다. 다음은 gzip 명령을 사용하는 방법에 대한 몇 가지 기본 예입니다.

- 파일을 압축하려면 다음 구문을 사용하십시오.

```
gzip file.txt
```

이렇게 하면 원본 파일의 압축 버전인 file.txt.gz라는 새 파일이 생성됩니다.

- 파일의 압축을 풀려면 다음 구문을 사용하십시오.

```
gzip -d file.txt.gz
```

이렇게 하면 파일의 압축이 풀리고 파일의 원래 버전인 file.txt라는 새 파일이 생성됩니다.

- 한 번에 여러 파일을 압축하려면 다음과 같은 와일드카드를 사용할 수 있습니다.

```
gzip *.txt
```

이렇게 하면 현재 디렉터리에서 확장자가 .txt인 모든 파일이 압축됩니다.

- 한 번에 여러 파일의 압축을 풀려면 다음과 같이 와일드카드를 사용할 수 있습니다.

```
gzip -d *.txt.gz
```

이것은 현재 디렉터리에서 확장자가 .txt.gz인 모든 파일의 압축을 풉니다.

- 파일을 압축하고 원본을 유지하려면 -k 또는 --keep 옵션을 사용할 수 있습니다.

```
gzip -k file.txt
```

- 압축 파일의 압축 비율과 원본 크기를 확인하려면 -l 또는 --list 옵션을 사용할 수 있습니다.

```
gzip -l file.txt.gz
```

위의 사용법은 gzip 명령으로 사용할 수 있는 많은 옵션의 몇 가지 예일 뿐입니다. 터미널에서 `man gzip`을 실행하여 gzip 매뉴얼 페이지에서 자세한 정보와 옵션을 찾을 수 있습니다.

다음은 실제 현장에서 자주 사용하는 파일 압축 및 해제하는 예 입니다.

```
# source directory 및 하위 파일을 하나의 아카이브 파일로 합친다.
tar -cvf archive.tar source

# 아카이브 파일을 압축 한다. 결과는 archive.tar.gz 파일로 생성되고 archive.tar 는 삭제
된다.
gzip archive.tar

# 복사한 .gz 파일의 압축을 해제 한다.
tar -xvf archive.tar.gz
```

7.2 심볼릭 링크

심볼릭 링크 또는 소프트 링크라고도 하는 심볼릭 링크는 다른 파일이나 디렉터리에 대한 참조 역할을 하는 Linux의 특수한 유형의 파일입니다. 심볼릭 링크는 Windows의 바로 가기 또는 MacOS의 alias 와 유사합니다. 심볼릭 링크에 액세스하면 운영 체제가 자동으로 심볼릭 링크가 가리키는 파일이나 디렉터리로 리디렉션합니다.

다음은 ln 명령을 사용하여 심볼릭 링크를 만드는 방법에 대한 몇 가지 기본 예입니다.

- 파일에 대한 심볼릭 링크를 만들려면 다음 구문을 사용하십시오.

```
ln -s /path/to/original/file /path/to/symlink
```

- 디렉토리에 대한 심볼릭 링크를 만들려면 다음 구문을 사용하십시오.

```
ln -s /path/to/original/directory /path/to/symlink
```

- symlink를 삭제하려면 unlink 명령을 사용하십시오.

```
unlink /path/to/symlink
```

- 심볼릭 링크 뒤의 실제 파일을 확인하려면 다음 명령을 사용하십시오.

```
ls -l /path/to/symlink
```

다음은 심볼릭 링크를 사용한 예입니다.

```
mkdir aaa
cd aaa
mkdir bbb
cd bbb
echo "cccccc" > ccc.txt

cd ../../
ln -s aaa/bbb/ccc.txt cct

ls -al
- 출력 -
lrwxrwxrwx. 1 testuser testuser 15 Jan 29 00:21 cct -> aaa/bbb/ccc.txt
```

```
cat cct
- 출력 -
CCCCCCCCCCCC
```

원본 파일에 직접 Access 하는 것과 동일한 결과가 나타납니다.

개발 단계 에서 Dynamic library 를 만들어 사용 할 경우 Version 별로 물리적인 파일이 생성 된 경우 테스트를 위해 심볼릭 링크를 이용하면 컴파일 환경(Makefile, ...)을 변경 하자 않고도 쉽게 테스트 할 수 있습니다.

7.3 notepad++ 에 FTP Plugin 설치 및 사용

출처: <http://triki.net/apps/3187>

Table of Contents

1. [NppFTP, FTP 플러그인](#)
2. [NppFTP 설치](#)
3. [FTP 창 활성화](#)
4. [NppFTP 기본 사용법](#)
5. [참고자료](#)

무료 소스 코드 프로그램인 Notepad++ 는 다양한 플러그인을 지원하지만 역시 그 중 최고는 NppFTP(FTP 플러그인) 입니다. 아래는 Notepad++ 에서 FTP 기능을 사용하기 위해 NppFTP 플러그인을 설치 하고 사용하는 기본 방법을 정리 한 것입니다.

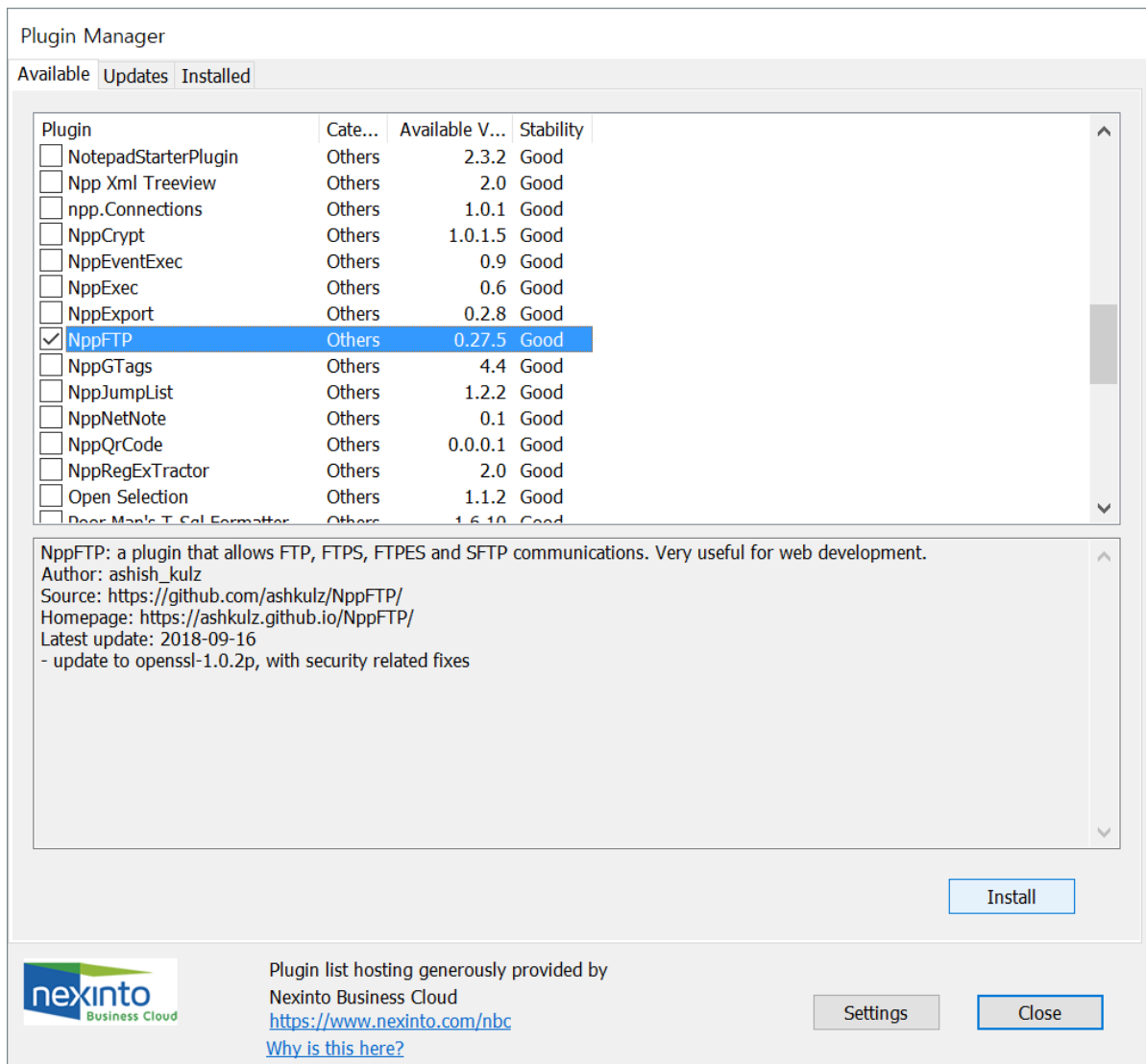
(아래 작업은 Notepad++ Plugin manager 가 설치 되어 있어야 합니다.)

NppFTP, FTP 플러그인

NppFTP 를 이용하면 아주 간단하게 자신의 사이트, 블로그의 소스 파일을 수정하고 업로드 할 수 있습니다.

NppFTP 설치

Notepad++ 를 열고 상단 메뉴에서 *Plugins - Plugin Manager - Show Plugin Manager* 로 이동합니다. Plugin Manager 창에서 NppFTP 를 찾아 체크하고 하단의 Install 버튼을 누르면 NppFTP 가 설치 됩니다.



Notepad++ 플러그인 매니저에서 NppFTP 설치

FTP 창 활성화

NppFTP 플러그인을 설치 했다면 NppFTP 창을 활성화 시켜 이용해야 합니다. 상단 메뉴의 *Plugins - NppFTP - Show NppFTP Window* 를 실행해 NppFTP 창을 활성화 시켜줍니다.

NppFTP 창이 활성화 되었다면, NppFTP 창 상단에서 제일 우측의 *Show messages window* 버튼을 눌러 NppFTP 메시지 창을 활성화 시킵니다. 이 창은 FTP 접속과 관련된 모든 로그를 보여 줍니다. 접속이 안 된 경우, 파일 업로드가 안된 경우 등을 확인 할 때 유용하므로 꼭 활성화 시켜주도록 합니다.

이제 본격적으로 Notepad++ 에서 FTP 기능을 사용할 수 있는 환경 구축이 완료 되었습니다.

NppFTP 기본 사용법

본격적으로 자신의 FTP 서버에 접속하기 위해서는 먼저 FTP 서버 프로필 세팅이 필요합니다. 활성화 시킨 NppFTP 창에서 상단 톱니바퀴 버튼(설정)을 눌러 *Profile settings* 메뉴로 들어 갑니다.

NppFTP – Profile settings 화면

새로운 창이 하나 켜지는데 여기서 *Add new* 버튼을 클릭해 자신이 관리하는 웹사이트의 FTP 서버에 대한 정보를 등록합니다.

프로필 세팅이 완료 되면 NppFTP Window 상단 제일 좌측의 연결 버튼이 활성화 됩니다. 연결 버튼을 이용해 자신의 FTP 서버에 접속한 후 본격적으로 파일 수정을 시작하면 됩니다.

파일 수정이 끝나고 저장만 하면 자동으로 서버에 파일이 업로드 됩니다.

조금은 복잡해 보이지만 사실은 간단한 이 과정이 Notepad++ 에서 NppFTP 플러그인 사용의 기본입니다.

참고자료

- [GitHub – NppFTP](#)

7.4 특정 패턴의 문자열이 포함된 파일 찾기

Linux 에서 개발 작업시 여러개의 소스 파일을 사용하여 개발을 진행 하게 됩니다. 이때 특정한 문자열 ("예: 함수") 이 포함되어 있는 파일이 어디에 있는지 기억이 나지 않을 경우 쉽게 찾을 수 있는 script 입니다.

사용법

```
`find . -name "파일 패턴" | xargs egrep "찾을 문자열"
```

ex)

```
find . -name "*.cpp" | xargs egrep "printf"
```

위 명령은 현재 경로 '.' 아래의 모든 경로에서 확장자가 cpp로 되어 있는 파일 중 printf 문자열을 가지고 있는 파일을 찾고 해당 내용을 console에 보여 달라는 명령입니다.

7.5 vi 편집기 사용하기

리눅스에서 가장 많이 사용되는 편집기는 vi 편집기입니다.

vi는 다른 편집기들과는 다르게 모드형 편집기라는 특징이 있습니다.

vi를 처음 기동시키면 명령 모드로 진입이 되며 이때는 글자를 입력할 수 없고 vi로 어떤 일을 하게될지 명령을 기다리는 상태가 됩니다.

vi의 모드는 명령 모드, 편집 모드, 라인 모드 3가지로 구분되며 각각의 모드별로 사용할 수 있는 기능은 다음과 같습니다.

명령 모드

- 모드 변경, 복사, 이동, 삭제, 등의 작업을 할 수 있습니다.
- 명령 모드에서 i, a, o 등의 키를 입력하면 편집 모드로 진입됩니다.

편집 모드

- 문서의 작성 및 편집을 할 수 있는 모드입니다.
- ESC 키의 입력에 의해 명령 모드로 복귀할 수 있습니다.

라인 모드 (또는 ex 모드)

- 라인 모드는 명령 모드에서 : key 입력에 의해 진입할 수 있습니다.
- :w, :q, :wq 등의 명령을 수행하여 쓰기 및 vi를 종료할 수 있습니다.
- :sh, bash 등 실행 명령을 통해 외부 파일을 실행할 수 있습니다.
 - 외부 파일의 실행이 종료되면 (sh, bash를 실행한 경우) exit 명령으로 다시 vi로 돌아올 수 있습니다.

표로 정리한 vi 모드

모 드	역할 및 특징
명 령 모 드	vi를 실행시키면 가장 먼저 접하는 기본이 되는 모드로 커서의 이동, 수정, 삭제, 복사 붙이기, 탐색 등을 한다. 입력 모드 전환키인 i,a,o,l,A,O 등을 입력하면 입력 모드로 전환되고, 명령 모드로 다시 전환하려면 [Esc] 키를 누르면 된다.
입 력 모 드	입력 모드 이외에도 편집 모드, input mode, insert mode 등으로 불리며, 글자를 입력하는 문서를 만드는 모드이다. 명령 모드에서 입력 전환키를 눌러서 전환하면 화면 아래에 '-- INSERT --'라고 표시된다.
ex 모 드	명령 모드에서 ':'키를 입력했을 때 화면 맨 아랫줄에서 명령을 수행하는 모드로 저장, 종료, 탐색, 치환 및 vi 환경 설정 등의 역할을 하는 모드이다.

입력 모드 전환 명령어

먼저 입력 모드로 들어가기 위해 i,a,o,l,A,O가 있다고 했는데 각각 무슨 차이일까

i	현재 커서의 위치부터 입력	I	현재 커서 줄의 맨 앞에서부터 입력 (shift + i)
a	현재 커서의 위치 다음 칸부터 입력	A	현재 커서 줄의 맨 마지막부터 입력 (shift + a)
o	현재 커서의 다음 줄에 입력	O	현재 커서의 이전 줄에 입력 (shfit + o)
s	현재 커서 위치의 한 글자를 지우고 입력	S	현재 커서의 한 줄을 지우고 입력 (shift + s)

커서 이동 명령어

사실 커서 이동은 키보드의 4개 화살표를 이용해서 이동하면 됩니다. 그러나 가끔 하위 버전의 Linux 또는 Unix 에서 vi 를 사용 할때 화살표 key 가 동작 하지 않을 때가 있습니다. 이 때는 hjkl key 를 이용하여 커서를 이동 해야 합니다.

명령	설명
h	커서를 왼쪽으로 한 칸 이동 (← 이랑 같은 의미)
j	커서를 아래로 한 칸 이동 (↓와 같은 의미)
k	커서를 위로 한 칸 이동 (↑와 같은 의미)
l	커서를 오른쪽으로 이동 (→와 같은 의미)
w	다음 단어의 처음으로 이동 (웬지 word의 약자 같은 느낌?)
^	줄의 첫 문자로 이동
\$	줄의 맨 끝으로 이동 (:\$은 줄의 제일 끝으로 이동)
0	첫 번째 열로 이동 (:0은 줄의 제일 처음으로 이동)
G	제일 끝 행으로 이동 (shift + G)
gg	제일 첫 행으로 이동 (g + g)
nG	n 번째 행으로 이동
:숫자 +enter	해당 숫자의 행으로 이동
H	화면의 첫 줄로 이동 (Head) (shift + h)
M	화면의 중간으로 이동 (Middle) (shift + m)
L	화면의 끝 줄로 이동 (Last) (shift + l)
[Ctrl] + b	이전 화면으로 이동 (Page Up과 같아요)
[Ctrl] + d]	반 전도 화면 이동 (스크롤 중간 정도 내린거 같은거.)
[Ctrl] + F	다음 화면으로 이동 (Page Down과 같아요)
n%	입력한 n퍼센트에 해당하는 줄로 이동

삭제 복사 붙여넣기 명령어

명령	설명
x	현재 커서가 위치한 문자를 삭제 (del와 같은 의미예요)
dw	단어 삭제
dd	현재 커서의 행 삭제
숫자 dd	현재 커서부터 숫자만큼의 행 삭제
yy	현재 커서가 있는 행을 복사
숫자 yy	현재 커서부터 숫자만큼의 행을 복사
p	복사한 내용을 현재 행 이후에 붙여 넣기
P	복사한 내용을 현재 행 이전에 붙여 넣기 (shift + p)

되돌리기 및 검색

명령	설명
u	직전에 내린 명령을 취소
/exp + enter	'exp' 와 같은 문자열을 현재 커서가 위치한 곳부터 아래 방향으로 검색
?exp + enter	'exp'와 같은 문자열을 뒤에서부터 찾습니다. 즉 위 방향으로 검색!
n	찾은 문자 중에서 다음 문자로 이동
N	n이 아래로 검색을 계속 내려가는 거라면 N은 위로 검색을 계속하는거 (shift + n)

vi 사용법 출처: <https://jhnyang.tistory.com/54>

vi 명령 모드에서 문자열 찾기

명령 모드 에서 / key 를 누르고 문자열을 입력 후 key 를 누르면 해당 문자열이 있는곳 으로 이동 합니다.

문자열을 발견 하지 못한 경우 'Pattern not found' 오류가 발생 합니다.

vi 명령 모드에서 문자열 일괄 수정 하기 찾기

명령 모드 에서 : key 를 눌러 라인 모드로 진입 후 다음과 같이 입력 하면 문자열을 바꿀 수 있습니다.

:start line num, end line num/원본문자열/바꿀문자열/g

ex)

```
:1,$s/read/write/g
```

설명: 해당 파일의 첫줄 부터 마지막 줄 까지 검색 하여 "read" 문자열을 "write" 로 수정 하라 는 명령 입니다.

7.6 script 명령어

Linux의 script 명령은 텍스트 파일의 모든 입력 및 출력을 캡처하여 터미널 세션의 typescript 를 만드는데 사용됩니다. script 는 exit 명령이 입력될 때까지 명령과 해당 명령의 출력을 포함하여 화면에 인쇄된 모든 것을 기록하는 것으로 시작합니다. 결과 typescript 는 나중에 저장하고 검토하거나 터미널에서 수행된 활동 로그로 사용할 수 있습니다. script 명령을 사용하기 위한 구문은 script [-a] [file]입니다. -a 옵션은 typescript를 덮어쓰는 대신 파일에 추가하며 file은 typescript를 저장할 파일의 이름입니다. 파일 이름을 지정하지 않으면 typescript는 typescript라는 파일에 저장됩니다.

7.6.1 리눅스 script 명령어 옵션

번호	옵션	long옵션	설명
1	-a	--append	이전에 작성되었던 script 내용에 추가하여 작성합니다.
2	-c	--command	지정한 특정 명령을 실행하여 파일명에 저장합니다.
3	-e	--return	자식 프로세스 exit code를 반환합니다.
4	-f	--flush	같은 서버의 접속한 다른사람에게 실시간으로 공유합니다.
5	-	--force	링크인 경우에도 출력파일로 사용합니다.
6	-q	--quiet	script 명령어 시작과 종료의 출력메세지를 출력하지 않습니다.
7	-t	--timing[=logfile]	script 명령어를 단계별로 캡처하여 재생할 수 있는 옵션입니다.
8	-V	--version	script 명령어의 버전을 출력합니다.
9	-h	--help	script 명령어의 사용법을 출력합니다.

script 명령은 타인이 내 터미널 에서 내리는 명령 및 화면 출력을 저장 하였다가 다시 확인 하고싶을 때 이용하면 좋습니다.

history 명령은 현재까지 수행했던 명령의 목록만 보여 주지만, script 명령은 출력 화면 까지 저장 해 주기 때문에 차이가 있습니다.

8. Linux kernel 및 Device Driver 소개

리눅스에서는 모든 것을 파일로 관리하게 되어있습니다.

어떤 장치를 리눅스에 마운트하면 리눅스에서는 해당 장치를 표현하는 장치 파일로 장치관리가 됩니다.

이 장치파일들을 사용할 수 있게 해주는 것이 바로 장치 드라이버(Device Driver)입니다.

8.1 device driver 구분 (블록 디바이스, 캐릭터 디바이스)

- 블록 디바이스(Block Device)

보통 하드디스크나 CD/DVD, 플로피디스크 등의 장치를 말하며, 블록이나 섹터 등의 정해진 단위로 데이터를 전송합니다. I/O 전송속도가 높은 것이 특징입니다.

- 캐릭터 디바이스(Character Device)

키보드, 마우스, 테이프, 모니터, 프린터 등의 장치가 있으며, byte 단위로 데이터를 전송합니다. I/O 전송속도가 다소 느릴 수도 있으나, 어플리케이션단에서 버퍼링을 제어하므로, 성능에 따라 차이가 있을 수 있습니다.

8.2 Kernel Version 확인 방법

Linux 에서 Device Driver 를 개발 할 경우 Kernel Version 에 따라 Device Driver 가 오동작 또는 Hang 이 걸리는 등 심각한 오류를 발생 시킬 수 있으므로 반드시 지원하는 Kernel Version 을 확인 후 kernel module 을 설치 하여야 합니다.

일반 개발자의 경우 Device Driver 를 개발할 가능성이 적지만 System Programming 을 하는 경우 Device Driver 를 이용해야 해결이 되는 경우가 있으므로 Kernel Version 은 매우 중요하게 확인 하여야 합니다.

Linux 의 kernel version 은 다음 명령을 실행 하여 알 수 있습니다.

```
[root@localhost ~]# uname -r
4.18.0-305.3.1.el8.x86_64
```

위 명령을 실행한 Linux 장치는 Kernel Version이 '4.18.0-305' 임을 알 수 있습니다. 대부분의 경우 4.18 까지만 확인이 되면 대부분 호환이 됩니다.

8.3 kernel Module 설치 방법

Linux Kernel 별로 지원 하는 함수의 prototype 이 수정/삭제 되는 경우가 있으므로 하위 Version 에서 지원하던 source 가 상위 version 에서 컴파일 되지 않는 경우가 자주 있습니다.

Linux 의 커널 모듈 설치에 별점 문서인 [리눅스 커널 컴파일\(Kernel Compile\) 및 모듈\(Module\) 관리.md](#) 파일을 참조 하시기 바랍니다.

9. 개발 Tool 설치 및 테스트 방법

9.1 yum 사용법

yum은 Red Hat 기반 Linux 배포판(예: Fedora 및 CentOS)용 패키지 관리자로 명령줄에서 패키지를 쉽게 설치, 업데이트 및 제거할 수 있습니다. yum을 사용하는 방법은 다음과 같습니다.

- 패키지 설치: yum install [패키지 이름]
- 패키지 업데이트: yum update [패키지 이름]
- 패키지 제거: yum remove [패키지 이름]
- 패키지 검색: yum search [keyword]
- 설치된 패키지 나열: yum list installed

- 사용 가능한 업데이트 나열: yum check-update
- yum 캐시 정리: yum clean all

참고: 충분한 권한을 얻으려면 sudo와 함께 yum 명령을 실행해야 할 수도 있습니다.

9.2 CentOS 8 미리 서버용 repo 파일 변경 방법

yum 을 이용하여 package 를 설치 할때 package 를 검색하는 경로인 repository 에 package 가 존재 하여야 검색 및 설치를 할 수 있습니다.

최근에 Red Hat 재단 에서 CentOS 의 지원을 하지 않기로 하면서 Package 보관하는 repository를 제대로 관리하지 않아 CentOS 의 package 설치시 오류가 발생 하는데 이때 대처 할 수 있는 방법 입니다.

root user 로 switch 후 작업을 진행 하여야 합니다.

```
cd /etc/yum.repos.d/

sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS-*

sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.org|g' /etc/yum.repos.d/CentOS-*
```

9.3 yum 을 이용한 gcc-c++ 설치

repository 의 변경 작업이 완료 되었으면 다음 명령을 실행 하여 c++ 컴파일러를 설치 할 수 있습니다.

```
yum install -y gcc-c++
```

다음 명령을 실행 하여 테스트를 진행 합니다.

```
[root@localhost ~]# c++
c++: fatal error: no input files
compilation terminated.
[root@localhost ~]#
```

10. Makefile 활용 방법

10.1 Makefile 이란 ?

Makefile은 소프트웨어 개발에서 프로젝트의 종속성을 빌드하고 관리하는 데 사용되는 특수 파일입니다. 프로젝트 빌드, 중간 파일 정리 및 테스트 실행과 같은 작업을 자동화하는 일련의 규칙을 제공합니다. Makefile에는 프로젝트 빌드 방법을 지정하는 대상, 종속성 및 명령 세트가 포함되어 있습니다. 'make' 유틸리티는 Makefile을 읽고 지정된 규칙에 따라 프로젝트를 빌드합니다.

* Windows 용 Visual C++ 에는 Makefile 파일을 이용하여 Build 하는 utility 로 nmake 가 있습니다.

10.2 자주 사용하는 자동변수

다음 표는 Makefile 생성시 자주 사용하는 자동 변수 입니다.

변수	설명
<code>\$@</code>	목표(출력, 생성되는) 이름
<code>\$*</code>	목표 이름에서 확장자가 없는 이름
<code>\$<</code>	첫 번째 전제 조건의 파일 이름
<code>\$?</code>	목표 파일 보다 더 최근에 갱신된 파일 이름
<code>^</code>	현재 Target이 의존하는 대상들의 전체 목록
<code>?</code>	현재 Target이 의존하는 대상들 중 변경된 것들의 목록

10.3 간단한 Makefile 예제 및 설명

```
# 컴파일러 지정
CC          = g++

# 최종 목적 파일
target      = TssNAS-Agent

# 임의의 변수 선언
tdx_dir     = ../tdxdisk_linux

# C 컴파일 옵션 지정
CFLAGS      = -m64 -fPIC -DMACHINE64 -D_FILE_OFFSET_BITS=64 -D_REENTRANT \
              -I. -I./include -I$(tdx_dir)/include -
I/usr/include/fuse

# C++ 컴파일 옵션 지정
CPPFLAGS    = $(CFLAGS) -std=c++11

# link 옵션 지정
LFLAGS      = -L. -L$(tdx_dir)/lib64 -lm -lrt -lfuse -lsecureCrypto \
              -ltdxdisk -ltdxapi -lpthread

LFLAGS_SSL  = -lssl -lcrypto

# source 파일 목록
SRCS        = main.cpp fs_usermode.cpp tss_utils.cpp \
              tss_crypto.cpp tss_acl.cpp

# object 목록을 만든다. SRCS 변수에서 .cpp 문자열을 .o 로 바꾼 목록을 만든다.
OBJS        = $(SRCS:.cpp=.o )

# 확장자가 .c 로 되어 있는 파일 컴파일 옵션
```

```

.C.O:
    $(CC) $(CFLAGS) -c $*.c

# 확장자가 .cpp 로 되어 있는 파일 컴파일 옵션
.cpp.o:
    $(CC) $(CXXFLAGS) -c $*.cpp

# 접미사(확장자) 종속성 옵션
.SUFFIXES: .o .c .cpp

# all 뒤의 종속 항목을 실행
all: $(OBJS) $(target) make_ac1 sha256hash

# target 변수에 선언된 목적물(실행파일 또는 object)이 OBJS 변수 목록에 선언된 파일보다 이
전 시간대에 생성이 되었으면
# 아래 명령을 실행 한다. $(CC) 부터 아래쪽의 종속된 명령은 반드시 TAB 으로 들여 쓰기가 되
어 있어야 한다.
$(target): $(OBJS)
    $(CC) -m64 -fPIC -o $@ $^ $(LFLAGS)
    cp -f $(target) ./bin/
    cp -f $(target) ../Installer/

make_ac1: make_ac1_main.cpp
    $(CC) -m64 -fPIC -o $@ $^
    cp -f $@ ./bin/
    cp -f $@ ../Installer/utils/

sha256hash: sha256hash.cpp
    $(CC) -m64 -fPIC -o $@ $^ $(LFLAGS_SSL)
    cp -f $@ ./bin/
    cp -f $@ ../Installer/utils/

# 중간 목적 파일 지우기
clean:
    rm -f *.o

# 중간 파일 및 최종 목적 파일 지우기
distclean:
    rm -f *.o
    rm -f $(target) make_ac1 sha256hash
    rm -f ./bin/$(target) ./bin/make_ac1 ./bin/sha256hash

# 최종 목적 파일 배포
dist:
    cp -f $(target) ./bin/
    cp -f make_ac1 ./bin/
    cp -f sha256hash ./bin/
    cp -f $(target) ../Installer/
    cp -f make_ac1 ../Installer/utils/
    cp -f sha256hash ../Installer/utils/

```

10.4 Static(정적 라이브러리) LIB 생성 및 활용

GCC(GNU Compiler Collection)는 정적 라이브러리와 동적 라이브러리를 모두 생성할 수 있습니다.

정적 라이브러리는 컴파일 시간에 실행 파일과 직접 연결됩니다. 결과 바이너리 파일에는 프로그램을 실행하는 데 필요한 모든 코드가 포함되어 있습니다. 이렇게 하면 결과 파일이 더 커지지만 자체 포함되어 있어 라이브러리를 설치하지 않고도 모든 시스템에서 실행할 수 있습니다.

공유 라이브러리라고도 하는 동적 라이브러리는 런타임에 연결됩니다. 실행 파일에는 라이브러리에 대한 참조만 포함되며 실제 코드는 별도의 파일에 저장됩니다. 이렇게 하면 실행 파일이 더 작아지지만 프로그램을 실행하려면 컴퓨터에 라이브러리를 설치해야 합니다. 동적 라이브러리는 프로그램을 다시 컴파일하지 않고도 업데이트할 수 있으므로 더 유연합니다.

GCC에서 -c 및 -ar 옵션을 사용하여 정적 라이브러리를 생성하고 -fPIC 및 -shared 옵션을 사용하여 동적 라이브러리를 생성할 수 있습니다.

다음은 테스트용 정적 라이브러리와 동적 라이브러리 생성에 사용되는 소스 파일들입니다.

subfunc.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void say_string(char* str)
{
    if(str == NULL)
    {
        printf("%s(%d) variable str is null\n", __FILE__, __LINE__ );
        exit(0);
    }
    printf(str);
}
```

subfunc.h

```
#include <stdio.h>

#ifndef SUBFUNC_H
#define SUBFUNC_H

void say_string(char* str);

#endif
```

main.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "subfunc.h"

int main(int argc, char** argv)
{
    say_string("hello world\n");
}

```

정적 라이브러리 생성용 Makefile (mk_static_lib.mk)

```

CC = gcc

target = libsubfunc.a

CFLAGS = -m64 -fPIC -c

SRCS = subfunc.c
OBJS = $(SRCS:.c=.o )

.c.o:
    $(CC) $(CFLAGS) -c $*.c

.SUFFIXES: .o .c

all: $(target)

$(target): $(OBJS)
    ar rvs $@ $(OBJS)
    rm -f $(OBJS)

clean:
    rm -f $(OBJS)

distclean:
    rm -f $(target)

```

실행 파일 생성용 Makefile

```

CC = gcc

target = testmain

CFLAGS = -m64 -fPIC -I. -DMACHINE64 -D_FILE_OFFSET_BITS=64 -D_REENTRANT

LFLAGS = -L. -lsubfunc

```



```

SRCS = main.c
OBJS = $(SRCS:.c=.o )

.c.o:
    $(CC) $(CFLAGS) -c $*.c

.SUFFIXES: .o .c

all: $(target)

$(target): $(OBJS)
    $(CC) -o $@ $(OBJS) $(LFLAGS)

clean:
    rm -f $(OBJS)

distclean:
    rm -f $(target)

```

위의 모든 파일을 같은 경로에 넣고 다음 명령을 실행 하여 컴파일을 실행 합니다.

```

[testuser@localhost StaticLIB]$ make -f mk_static_lib.mk
gcc -m64 -fPIC -c -c subfunc.c
ar rvs libsubfunc.a subfunc.o
r - subfunc.o
rm -f subfunc.o

[testuser@localhost StaticLIB]$
[testuser@localhost StaticLIB]$ make all
gcc -o testmain main.o -L. -lsubfunc
[testuser@localhost StaticLIB]$

```

make utility 는 옵션 없이 실행 하면 Makefile 을 찾아서 컴파일을 실행 합니다. 그러나 Makefile 대신 다른 이름의 파일을 이용하여 컴파일 할 때는 -f 옵션을 이용하여 컴파일 하면 됩니다.

위 컴파일 결과 실행 파일 testmain 이 생성 되었고 ./testmain 으로 실행 하면 "hello world" message 를 표시하며 프로그램이 종료 됩니다.

10.5 Dynamic(동적 라이브러리) LIB 생성 및 활용

동적 라이브러리 생성에 사용하는 source(subfunc.c, subfunc.h, main.c) 파일은 위의 정적 library 를 테스트 할때 사용한 source 를 그대로 사용 합니다.

아래는 동적 라이브러리를 만들고 사용 할때 쓰는 Makefile 입니다.

동적 라이브러리 생성용 Makefile (mk_so.mk)

```

CC = gcc

target = libsubfunc.so

```

```

CFLAGS = -m64 -fPIC -c
LFLAGS = -shared

SRCS = subfunc.c
OBJS = $(SRCS:.c=.o )

.c.o:
    $(CC) $(CFLAGS) -c $*.c

.SUFFIXES: .o .c

all: $(target)

$(target): $(OBJS)
    $(CC) -o $@ $(OBJS) $(LFLAGS)
    rm -f $(OBJS)

clean:
    rm -f $(OBJS)

distclean:
    rm -f $(target)

```

정적 라이브러리용 Makefile 과 유사 하지만 target 과 LFLAGS 가 바뀐것에 주목 하시기 바랍니다.

실행 파일 생성용 Makefile

```

CC = gcc

target = testmain

CFLAGS = -m64 -fPIC -I. -DMACHINE64 -D_FILE_OFFSET_BITS=64 -D_REENTRANT

LFLAGS = -L. -lsubfunc

SRCS = main.c
OBJS = $(SRCS:.c=.o )

.c.o:
    $(CC) $(CFLAGS) -c $*.c

.SUFFIXES: .o .c

all: $(target)

$(target): $(OBJS)
    $(CC) -o $@ $(OBJS) $(LFLAGS)

clean:

```

```
rm -f $(OBJ)
```

```
distclean:
```

Makefile 은 정적 컴파일때 사용한것을 그대로 사용 합니다.

위의 모든 파일을 같은 경로에 넣고 다음 명령을 실행 하여 컴파일을 실행 합니다.

```
[testuser@localhost SharedOBJ]$ make -f mk_so.mk
gcc -m64 -fPIC -c -c subfunc.c
gcc -o libsubfunc.so subfunc.o -shared
rm -f subfunc.o
```

```
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$ make
gcc -o testmain main.o -L. -lsubfunc
[testuser@localhost SharedOBJ]$
```

컴파일이 정상적으로 수행 되었습니다.

하지만 아래 명령으로 실행을 해 보면 오류가 발생 합니다.

```
[testuser@localhost SharedOBJ]$ ./testmain
./testmain: error while loading shared libraries: libsubfunc.so: cannot open
shared object file: No such file or directory
```

같은 경로에 있는 libsubfunc.so 파일을 찾을 수 없다는 오류 입니다.

위 오류를 다시 확인 하기 위해서 다음 명령을 실행 합니다.

```
[testuser@localhost SharedOBJ]$ ldd ./testmain
linux-vdso.so.1 (0x00007fff6fb32000)
libsubfunc.so => not found
libc.so.6 => /lib64/libc.so.6 (0x00007f16a2f89000)
/lib64/ld-linux-x86-64.so.2 (0x00007f16a334e000)
[testuser@localhost SharedOBJ]$
```

./testmain 를 실행 하기 위해서 필요한 .so 파일들과 해당파일의 경로 및 파일명이 표시 되는데 "libsubfunc.so => not found" 즉 libsubfunc.so 파일을 찾지 못하고 있습니다.

해당 오류를 해결 하기 위해서 다음 명령을 실행 합니다.

```
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$ export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$ ldd ./testmain
        linux-vdso.so.1 (0x00007ffc143e6000)
        libsubfunc.so => ./libsubfunc.so (0x00007fd8a3e22000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fd8a3a5d000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fd8a4024000)
[testuser@localhost SharedOBJ]$
```

위 명령 실행 후 다시 ./testmain 을 실행 하면 정상적으로 동작 합니다.

```
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$ ./testmain
hello world
[testuser@localhost SharedOBJ]$
```

10.6 기타

Makefile 여러 경로에 있는 소스 일괄 Build 는 별도로 첨부된 [Makefile 여러 경로에 있는 소스 일괄 Build](#) 를 참조 하세요

Makefile 에서 Shell 사용하기 는 별도로 첨부된 [Makefile 에서 Shell 사용하기](#) 를 참조 하세요.

- Linux C/C++ 에서 Daemon 프로그램 만들기는 [MakeDaemon.md](#) 을 참조 하세요

10.7 C/C++ 개발 시 참조용 명령어

다음 명령어(tool) 들은 C/C++ 로 App 개발시 유용하게 사용 할 수 있는 명령어 들 입니다.

10.7.1 ldd

Linux의 'ldd' 명령은 주어진 실행 파일에 필요한 공유 라이브러리를 인쇄하는 데 사용됩니다. 관련 버전 정보와 함께 실행 파일이 의존하는 공유 라이브러리의 이름과 경로를 표시합니다.

'ldd' 명령의 기본 구문은 다음과 같습니다.

```
ldd [실행파일명]
```

예를 들어 'ls' 명령에 필요한 공유 라이브러리를 보려면 다음 명령을 실행합니다.

```
ldd /bin/ls
```

'ldd' 명령의 출력은 다른 시스템에서 실행 파일을 실행할 때 누락된 라이브러리 종속성 또는 호환성 문제를 해결하는 데 유용할 수 있습니다.

10.7.2 nm

Linux의 'nm' 명령은 개체 파일 또는 실행 파일의 기호 테이블을 표시하는 데 사용됩니다. 기호 테이블은 프로그램에서 사용되는 기호 목록과 기호의 주소, 크기 및 유형과 같은 관련 정보입니다.

'nm' 명령의 기본 구문은 다음과 같습니다.

```
nm [object파일명]
```

예를 들어 개체 파일 'main.o'의 기호 테이블을 표시하려면 다음 명령을 실행합니다.

```
nm main.o
```

'nm' 명령의 출력에는 기호 유형(예: 전역 함수의 경우 T, 전역 변수의 경우 D), 크기 및 주소와 같은 관련 정보와 개체 파일의 기호 목록이 표시됩니다.

'nm' 명령은 종종 프로그래머가 개체 파일에 정의된 기호를 확인하거나 공유 라이브러리에서 사용되는 기호를 검사하는 데 사용됩니다. 누락된 기호 또는 기호 해결과 관련된 문제를 디버깅하는 데에도 사용할 수 있습니다.

11. 개발 능률 향상을 위해 vi 에 설치하는 Plugin

Linux 에서 C/C++ 개발시 능률 향상을 위해 ctags 및 taglist 를 설치하여 사용하는 방법 입니다.

11.1 ctags 설치 및 사용

ctags는 소스 코드의 Symbol(전역변수, 함수, 매크로 정의 등)들의 정보를 모아 tags 파일을 생성하는 Tool 입니다.

주로 소스 분석 시 사용하는데 vi(vim)과 함께 사용 시 간단하고 직관적인 인터페이스로 굉장히 유용하게 사용됩니다.

사용법 자체는 매우 간단합니다.

11.1.1 설치

아래 명령을 실행 하여 ctags 를 설치 합니다.

```
yum install -y ctags
```

11.1.2 tags 생성

분석하고자 하는 소스의 최상위 디렉토리로 이동한 뒤 다음 명령어를 실행하여 tags 정보를 생성합니다.

```
ctags -R
```

11.1.3 vi 에 tags 경로 지정

다음 명령을 실행 하여 vi(vim, 이하 vi) 에 tags 위치를 지정 합니다.

```
vi ~/.vimrc

# vi 가 기동 되면
set tags=/home/testuser/TEST_DIR/tags
```

11.1.4 심볼(함수, 전역변수, ...) 이 정의된 곳으로 이동 및 복귀

vi 를 이용하여 소스 파일을 열고 보던 중 함수의 원형을 보려고 하면 해당 함수의 위치로 이동 후 ctrl 키 + 우측 대괄호 key(**ctrl +]**) 를 눌러서 해당 함수로 이동 합니다.

ctrl +] 를 눌러 이동 하여 함수를 살핀 후 원래 보던 source 쪽으로 이동을 하려면 **ctrl + t** key를 눌러 복귀 합니다.

11.2 taglist 설치 및 사용

11.2.1 설치

Taglist는 소스 코드 탐색을 위한 vim 플러그인입니다. 사용하려면 Vim이 설치되어 있어야 합니다. 다음은 Taglist 사용 방법에 대한 기본 가이드입니다.

- 플러그인 설치: <https://sourceforge.net/projects/vim-taglist/files/latest/download> 에서 taglist_46.zip 파일을 download 합니다.
 - 해당 계정에만 적용 방법
다음 명령을 실행 합니다.

```
mkdir ~/.vim

cp taglist_46.zip ~/.vim

cd ~/.vim

unzip taglist_46.zip
```

- 전체 계정에 적용 하는 방법
 - 압축을 해제 하여 doc/taglist.txt 파일을 `/usr/share/vim/vim80/doc` 에 복사 합니다.
 - 압축을 해제 하여 plugin/taglist.vim 파일을 `/usr/share/vim/vim80/plugin` 에 복사 합니다.

이제 설치는 완료 되었습니다.

- 태그 파일 생성: 소스 코드 디렉토리에서 ctags 명령을 실행하여 태그 파일을 생성합니다.
- Vim 열기: Vim에서 검색하려는 파일을 엽니다.
- 태그 목록 호출: vi 의 명령 모드 에서 :Tlist를 입력하여 태그 목록 창을 엽니다.
- 태그 찾아보기: 커서 키를 사용하여 태그를 탐색하고 Enter 키를 눌러 선택한 태그로 이동합니다.
- 태그 목록 닫기: :q를 입력하여 태그 목록 창을 닫습니다.

11.2.2 사용

이제 vi 에서 원하는 소스 파일을 열고 명령 모드 에서 `:Tlist` 를 입력 하면 tags 목록 창이 왼쪽에 나타납니다.

11.2.3 taglist 사용 참조

설정

1. taglist 다운(<http://vim-taglist.sourceforge.net/>)
2. mkdir ~/.vim
3. cp taglist_46.zip ~/.vim
4. cd ~/.vim
5. unzip taglist_46.zip

- 사용

1. vi 에서 실행
:Tlist

2. 좌우 이동

Ctrl + w + w

3. 태그창에서 분류 접었다 펴기

+/-

4. .vimrc 설정

- 태그 범위(함수, 매크로, 구조체 등)를 표시

```
let Tlist_Display_Tag_Scope = 1
```

- 함수 원형을 표시

```
let Tlist_Display_Prototype = 1
```

- 태그 리스트 소팅 (소스 코드 위치 순서가 아닌 이름 순서로 표시)

```
let Tlist_Sort_Type = "name"
```

- 태그 리스트 창을 우측에 표시

```
let Tlist_Use_Right_Window = 1
```

- 태그 리스트 창의 폭을 35문자로 지정

```
let Tlist_winwidth = 35
```

12. 기타 Tool 및 참조 할 만한 기술 소개

12.1 scp 사용법

scp 는 ssh 원격 접속 프로토콜을 기반으로 한 SecureCopy(scp)의 약자로서 원격지에 있는 파일과 디렉터리를 보내거나 가져올 때 사용하는 파일 전송 프로토콜입니다. 네트워크가 연결되어 있는 환경에서 ssh와 동일한 22번 포트와 identity file을 사용해서 파일을 송수신하기 때문에 보안적으로도 안정된 프로토콜이라고 할 수 있습니다. 사용법은 아래와 같습니다.

12.1.1 단일 파일을 원격지로 보낼 때.

scp [옵션] [파일명] [원격지id]@[원격지ip]:[받는 위치]

```
scp testfile2 root@192.168.198.132:/tmp/datadir
```

현재 위치의 testfile2를 원격지 192.168.198.132:/tmp/datadir경로에 파일을 전송합니다.

12.1.2 복수의 파일을 원격지로 보낼 때.

구문 : # scp [옵션] [파일명 1] [파일명 2] [원격지id]@[원격지ip]:[받는 위치]

```
scp tesfile1 testfile2 root@192.168.198.132:/tmp/datadir
```

현재 위치의 tesfile1 testfile3을 동시에 원격지 192.168.198.132:/tmp/datadir경로에 파일을 전송합니다.

12.1.3 여러 파일을 포함하고 있는 디렉터리를 원격지로 보낼 때

구문 : # scp [옵션] [디렉터리 이름] [원격지id]@[원격지ip]:[보낼 경로]

```
scp -r testdata root@192.168.198.132:/tmp/datadir
```

옵션으로 -r 을 사용 합니다.

Linux 에서 -r 또는 -R 을 옵션으로 사용하는 경우는 대부분 recursive(재귀)를 의미 합니다. (예: sub directory 전체가 대상임)

현재 위치의 testdata directory 아래에 있는 모든 파일과 directory 그리고 subdirectory 에 있는 모든 파일 들을 root@192.168.198.132:/tmp/datadir 경로로 전송 합니다.

추가적인 옵션을 살펴보면 다음과 같습니다.

옵션	내용	사용 예
r	디렉토리 내 모든 파일/디렉토리 복사	scp -r
p (소문자)	원본 권한 속성 유지 복사	scp -p
P (대문자)	포트 번호 지정 복사	scp -P [포트번호]
c (소문자)	압축 복사	scp -c
v	과정 출력 복사	scp -v
a	아카이브 모드 복사	scp -a

12.2 네트워크 모니터링 (netstat)

Linux의 'netstat' 명령은 네트워크 연결, 라우팅 테이블 및 네트워크 통계와 같은 네트워크 관련 정보를 표시하는 데 사용됩니다.

'netstat' 명령의 기본 구문은 다음과 같습니다.

```
netstat [options]
```

'netstat' 명령에 일반적으로 사용되는 몇 가지 옵션은 다음과 같습니다.

- -a: 모든 소켓 표시(수신 및 설정된 연결 모두)
- -t: TCP 연결만 표시
- -u: UDP 연결만 표시

- `-l`: listening소켓만 표시
- `-n`: 호스트 이름을 확인하는 대신 숫자 주소 표시
- `-p`: 열린 port 번호를 표시 합니다.

아래는 자주 사용하는 option 을 활용한 예 입니다.

```
[testuser@localhost SharedOBJ]$
[testuser@localhost SharedOBJ]$ netstat -antp | grep 22
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 192.168.122.1:53      0.0.0.0:*              LISTEN
-
tcp        0      0 0.0.0.0:22            0.0.0.0:*              LISTEN
-
tcp        0      0 192.168.198.132:22    192.168.198.1:60229    ESTABLISHED
-
tcp        0      0 192.168.198.132:22    192.168.198.1:65425    ESTABLISHED
-
tcp        0      0 192.168.198.132:22    192.168.198.1:65282    ESTABLISHED
-
tcp        0      0 192.168.198.132:22    192.168.198.1:65424    ESTABLISHED
-
tcp        0      48 192.168.198.132:22    192.168.198.1:60226    ESTABLISHED
-
tcp        0      0 192.168.198.132:22    192.168.198.1:65281    ESTABLISHED
-
tcp6       0      0 :::22                 :::*                    LISTEN
-
tcp6       0      0 :::1:53322            :::1:6011               ESTABLISHED
4439/dbus-launch
tcp6       0      0 :::1:6011              :::1:53322              ESTABLISHED
-
[testuser@localhost SharedOBJ]$
```

22번 포트가 열려 있는걸 확인 할때 사용 합니다.

실제 Linux 에서 port 를 통해 전송되는 Data 를 모니터링 할 경우는 Wireshark 를 많이 이용합니다

12.3 내가 작성한 함수를 대신 호출 해줘(LD_PRELOAD)

LD_PRELOAD는 프로그램이 실행될 때 다른 라이브러리보다 먼저 로드되어야 하는 추가 사용자 정의 공유 라이브러리를 지정하는 데 사용되는 Linux의 환경 변수입니다. LD_PRELOAD로 지정된 라이브러리는 표준 시스템 라이브러리보다 먼저 로드되어 사용자가 표준 라이브러리 기능의 동작을 무시하거나 시스템 라이브러리에서 제공하지 않는 추가 기능을 로드할 수 있습니다.

LD_PRELOAD를 사용하기 위한 기본 구문은 다음과 같습니다.

```
LD_PRELOAD=[library_path1] [library_path2] ... [program_name]
```

예를 들어, 다른 라이브러리보다 먼저 로드된 사용자 지정 라이브러리 'my_library.so'가 있는 프로그램 'my_program'을 실행하려면 다음 명령을 실행합니다.

```
LD_PRELOAD=./my_library.so ./my_program
```

LD_PRELOAD는 사용자 지정 라이브러리 디버깅, 테스트 및 개발에 자주 사용되며 성능 또는 호환성 이유로 시스템 라이브러리의 동작을 재정의하는 데 사용할 수도 있습니다. 그러나 LD_PRELOAD를 잘못 사용하면 예측할 수 없는 동작 및 안정성 문제가 발생할 수 있으므로 주의해서 사용해야 합니다.

13. 별도 교육 요청 사항 및 추천 도서

13.1 C의 system 함수 사용 방법

gcc의 system() 함수는 shell 명령을 실행 하는 함수 입니다. App 에서 system() 함수를 호출 하면 해당 함수가 종료 될 때 까지 App 이 멈추어 있게 됩니다.

system 함수의 prototype 은 `int system(const char *command);` 입니다, command 에 실행할 명령을 기술 하면 됩니다.

system() 함수는 실행에 실패 하면 -1 값을 반환 하며, 성공 하였을 경우 해당 명령의 종료 값 을 반환 합니다. (해당 명령의 exit(값))

system() 함수를 중단 없이 사용 하고자 할 경우 fork() 함수를 사용하여 child process 를 생성 한 후 호출 하는 방법이 유용 하며, system() 함수와 유사한 함수들 에는 execv() 등이 있습니다.

다른 process 에서 출력 하는 여러 줄의 message 를 받아서 처리할 경우 또는 다른 process 와 상호 작용할 필요가 있을 경우에는 popen() 함수의 사용을 고려 해야 합니다.

아래는 system() 함수 사용 예 입니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char** argv)
{
    char buff[ 256 ];
    int nres = 0;

    memset(buff, 0x00, sizeof(buff));
    sprintf(buff, "%s -a1tr");

    nres = system(buff);
    exit(0);
}
```

C/CPP 에서 중단 없이 외부 프로그램 실행 시키는 방법은 별도로 첨부된 [C에서 외부 프로그램 실행 시키기.md](#) 파일을 참조 하세요.

13.2 RTC 설정 및 읽는 방법

13.2.1 RTC 설정 방법

Linux 시스템의 실시간 시계(RTC)는 `hwclock` 명령을 사용하여 설정할 수 있습니다. `hwclock` 명령은 명령줄에서 하드웨어 시계(BIOS 또는 CMOS 시계라고도 함)를 읽고 설정하는 데 사용됩니다. Linux에서 RTC를 설정하는 구문은 다음과 같습니다.

```
hwclock --set --date "MM/DD/YYYY HH:MM:SS"
```

이 명령은 하드웨어 시계를 지정된 날짜 및 시간으로 설정합니다. 날짜 및 시간은 'MM/DD/YYYY HH:MM:SS'(월/일/년 시:분:초) 형식이어야 합니다.

--systohc 옵션을 사용하여 하드웨어 시계를 현재 시스템 시간으로 설정할 수도 있습니다.

```
hwclock --systohc
```

하드웨어 시계가 잘못된 시간으로 설정된 경우, 특히 시스템의 시간대가 올바르게 설정되지 않은 경우 문제가 발생할 수 있으므로 `hwclock` 명령을 주의해서 사용해야 한다는 점도 중요합니다.

13.2.2 RTC 읽는 방법

Linux에서 RTC(Real-Time Clock)는 `ioctl` 시스템 호출 및 `RTC_RD_TIME` 요청을 사용하여 C에서 읽을 수 있습니다.

다음은 C에서 RTC 시간을 읽는 방법의 예입니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/rtc.h>
#include <sys/ioctl.h>

int main() {
    int rtc_fd;
    struct rtc_time rtc_tm;

    rtc_fd = open("/dev/rtc0", O_RDONLY);
    if (rtc_fd == -1) {
        perror("open");
        exit(1);
    }

    if (ioctl(rtc_fd, RTC_RD_TIME, &rtc_tm) == -1) {
        perror("ioctl");
        close(rtc_fd);
        exit(1);
    }

    printf("Current RTC date/time is %d-%d-%d, %02d:%02d:%02d.\n",
        rtc_tm.tm_mday, rtc_tm.tm_mon + 1, rtc_tm.tm_year + 1900,
        rtc_tm.tm_hour, rtc_tm.tm_min, rtc_tm.tm_sec);
}
```

```

close(rtc_fd);
return 0;
}

```

이 코드는 open() 함수를 사용하여 '/dev/rtc0'에 있는 RTC 장치 파일을 연 다음 RTC_RD_TIME 요청과 함께 ioctl() 함수를 사용하여 현재 RTC 시간을 rtc_time 구조체로 읽습니다. 구조체에는 일, 월, 연도, 시, 분 및 초에 대한 필드가 포함되어 있습니다. 그런 다음 날짜와 시간이 콘솔에 인쇄됩니다.

RTC 장치 파일은 '/dev/rtc' 또는 '/dev/misc/rtc'와 같이 시스템의 다른 경로에 있을 수 있습니다. 또한 이 예제는 RTC 장치 드라이버용으로 작성되었으며 다른 장치의 경우 접근 방식이 다를 수 있습니다.

13.3 C에서 Linux 작업 스케줄링

Linux에서 C의 태스크 스케줄링은 sched.h 라이브러리의 sched_setscheduler() 함수를 사용하여 달성할 수 있습니다. 이 기능은 주어진 프로세스에 대한 스케줄링 정책과 우선 순위를 설정합니다.

다음은 C에서 프로세스에 대한 스케줄링 정책 및 우선 순위를 설정하는 방법의 예입니다.

```

#include <sched.h>
#include <unistd.h>

int main() {
    int policy = SCHED_FIFO; // or SCHED_RR, SCHED_OTHER
    struct sched_param param;
    param.sched_priority = 1; // set priority to 1

    int ret = sched_setscheduler(0, policy, &param);
    if (ret == -1) {
        perror("sched_setscheduler");
        exit(1);
    }
    // 나머지 코드
    return 0;
}

```

이 코드는 우선 순위가 1인 First-In First-Out 실시간 스케줄링을 나타내는 SCHED_FIFO로 스케줄링 정책을 설정합니다. 다른 가능한 정책으로는 라운드 로빈 스케줄링을 위한 SCHED_RR과 표준 Linux 스케줄러를 위한 SCHED_OTHER가 있습니다. sched_setscheduler()의 첫 번째 매개변수는 프로세스 ID이며, 이 경우 0은 현재 프로세스를 의미합니다.

sched_param 구조체의 sched_priority 필드는 프로세스의 우선 순위를 설정합니다. 유효한 우선 순위 범위는 일정 정책에 따라 다르며 SCHED_FIFO 및 SCHED_RR의 경우 0

99, SCHED_OTHER의 경우 0

sched_get_priority_max(SCHED_OTHER)일 수 있습니다.

높은 우선 순위를 설정한다고 해서 프로세스가 항상 실행된다는 보장은 없으며 시스템 부하 및 기타 요인에 따라 달라집니다. 또한 실시간 스케줄링 정책은 적절하게 사용되지 않으면 시스템 전체 잠금을 유발할 수 있으므로 위험할 수 있습니다.

프로세스에 대한 스케줄링 정책 및 우선순위를 설정하려면 수퍼유저 권한이 필요할 수 있으며 스케줄링 정책이 성공적으로 설정되었는지 확인하려면 sched_setscheduler() 함수의 반환 값을 확인해야 합니다.

13.4 추천 도서

- 오라일리 재팬 에서 발행한 'binary hacks' (한글판, 알라딘)
- 실용주의 프로그래머

그리고 대형 Project 를 수행 할 때 OOP 를 적용 할 경우 **디자인 패턴** 관련 공부와 **개발 방법론** 관련 공부도 해두면 좋습니다.

14. 실습 및 Q&A
