

Linux 운영체제 운설습

Section 1.

Linux Setup

지금부터 우리는 리눅스를 설치합니다. 리눅스를 설치하는 방법은 다양하지만 여기에서는 Oracle의 가상 머신 프로그램인 Virtual Box를 이용하여 가상으로 리눅스를 설치하는 방법에 대해 여러분들에게 설명합니다.



Supplies

Virtual Box

Oracle에서 만든 가상 시스템 프로그램으로, 운영체제 내부에 하나의 버추얼 컴퓨터를 만들어줍니다. <https://www.virtualbox.org/>에서 각각의 운영체제에 맞는 최신 버전의 Virtual Box 프로그램을 다운받습니다.

Xubuntu

Xubuntu는 가장 많은 사람들이 사용하는 리눅스 배포판인 Ubuntu의 여러 종류 중 하나입니다. 교재에서는 정식으로 xubuntu 13.04버전을 기준으로 사용합니다. Ubuntu에서 사용하는 Unity 인터페이스를 버리고 Gnome3과 xfce 인터페이스를 차용하여 가벼운 것이 특징입니다. 이 버전의 경우 <http://www.xubuntu.org/>에서 받을 수 있습니다.

Lecture

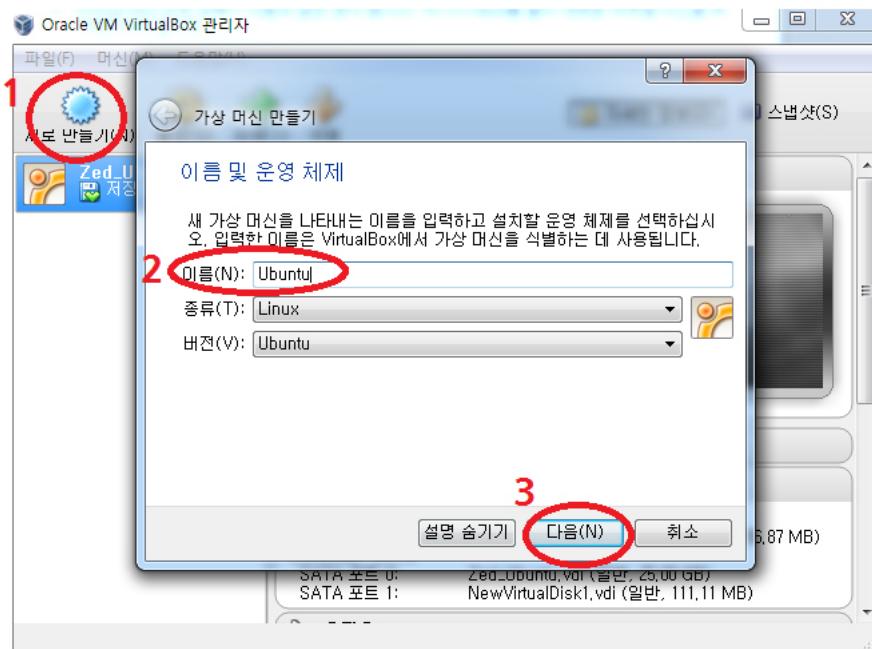
이번 강의에서는 컴퓨터에 리눅스를 설치합니다.

설치는 크게 다음과 같은 순서로 진행됩니다.

1. VirtualBox 설치
2. Xubuntu 13.04 설치
3. 소프트웨어 저장소를 다음(ftp.daum.net)으로 변경
4. 저장소 목록 업데이트 / 패키지 업그레이드
5. Guest 플러그인 설치
6. 사용언어 한글로 변경

1. Virtual Box 최초 실행시 다음과 같은 창이 뜹니다. 여기서 [새로 만들기] 버튼을 눌러 새로운 버추얼 머신을 추가해주어야 합니다.

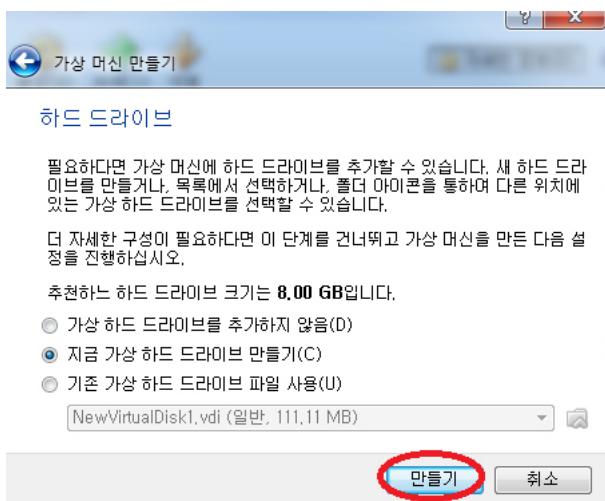
새로 버추얼 머신을 만들 경우 이름을 입력하는 창이 뜨게 되는데, 여기서 Xubuntu를 입력하면 자동으로 운영체제를 인식하여 종류와 버전이 자동으로 세팅됩니다. 각각 Linux와 Ubuntu로 입력되었는지 확인한 후에 다음 버튼을 눌러 다음 단계로 넘어갑니다.



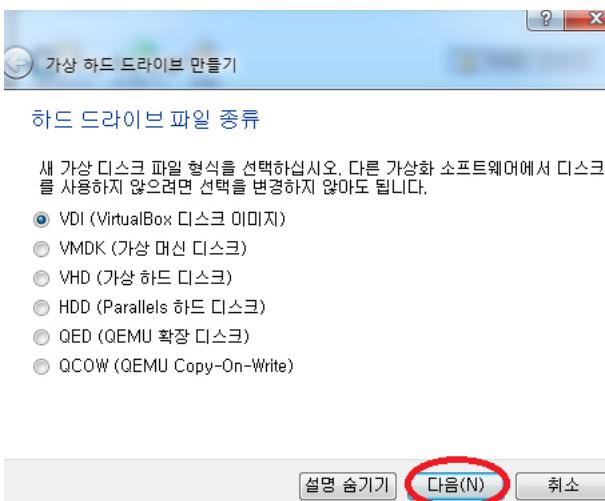
2. 메모리와 하드 드라이브를 설정해 줍니다. 수업에서는 공식적으로 램 1기가와 15GB의 고정 하드 드라이브 용량을 사용합니다.



수업에서 사용하는 가상 머신은 기본적으로 1기가 정도의 메모리를 가지고 있는 것이 좋습니다. 그렇지만 컴퓨터 환경에 따라 적절히 조정하는 것도 괜찮습니다. 메모리 용량을 설정하고 다음 버튼을 눌러줍니다.



다음으로 하드 드라이브를 설정하는 메뉴가 나오는데, 리눅스를 설치할 공간은 반드시 필요하므로 지금 가상 하드 드라이브 만들기를 클릭해서 드라이브 공간을 만들어 주어야 합니다.



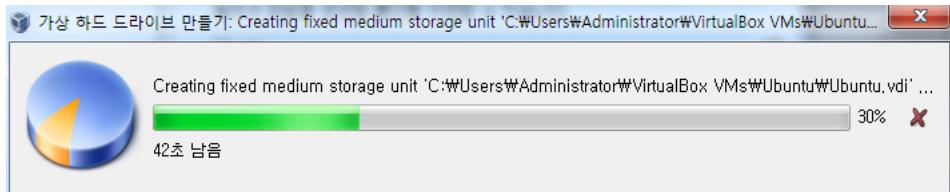
하드 드라이브 파일 종류는 굉장히 많이 있지만 여기서는 기본적인 VDI를 지정해 줍니다. VDI는 Virtual Box의 표준 드라이브 형태로, 하나의 단일 파일을 만들어서 하드 디스크와 같은 형태로 사용하는 방식입니다.

이렇게 설정을 한 뒤에는 하드디스크의 크기를 가변적인 동적 할당을 할 것인지, 아니면 고정된 크기로 지정할 것인지를 정해 주어야 합니다.

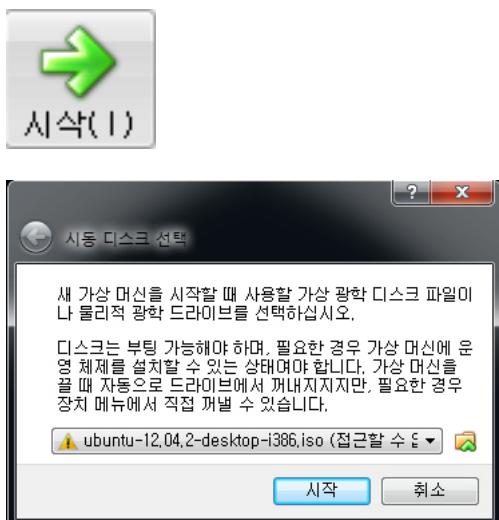
- 동적 할당(D)
- 고정 크기(F)

SSD를 사용하고 있다면 접근 속도가 빠르기 때문에 동적 할당을 해주어도 좋지만, 여기서는 미리 15GB의 넉넉한 테스트 공간을 지정해주도록 하겠습니다.

3. 설정을 마무리하는 단계입니다. 아래와 같은 가상 드라이브를 만드는 순서 이후, 가상 컴퓨터가 만들어지고, 그것을 실행하면서 본격적으로 우분투 리눅스를 설치할 준비를 합니다.



이제 리눅스를 설치하기 위해 컴퓨터를 구동해야 합니다. 만들어진 가상 머신을 클릭하고, 프로그램 상단의 [시작] 버튼을 눌러 가상 컴퓨터를 시동시킵니다.



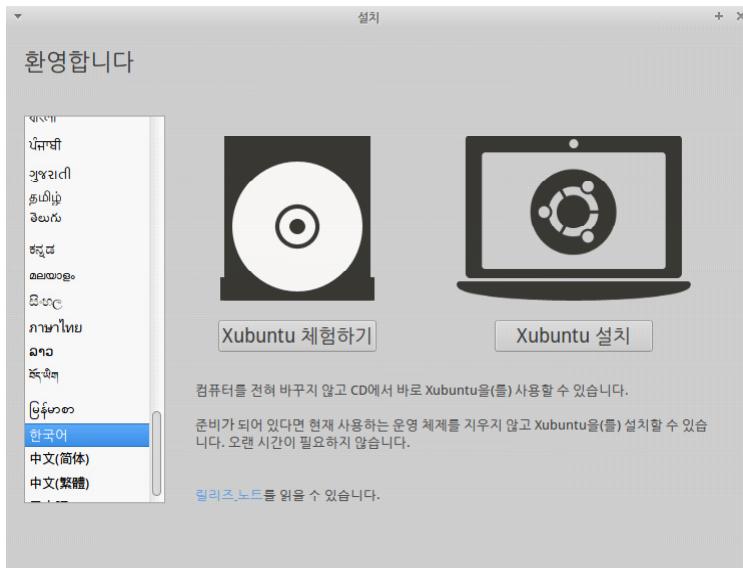
컴퓨터를 처음 시동할 경우 다음과 같은 창이 뜹니다. 이는 우리가 컴퓨터를 윈도우가 깔려있지 않은 상태에서 삼을 때 윈도우를 설치하기 위해 CD를 넣으라는 것과 같은 의미입니다. 본 강의를 시작하기 앞서 받아놓은 우분투 이미지 경로를 입력하고 [시작] 버튼을 눌러 줍니다.



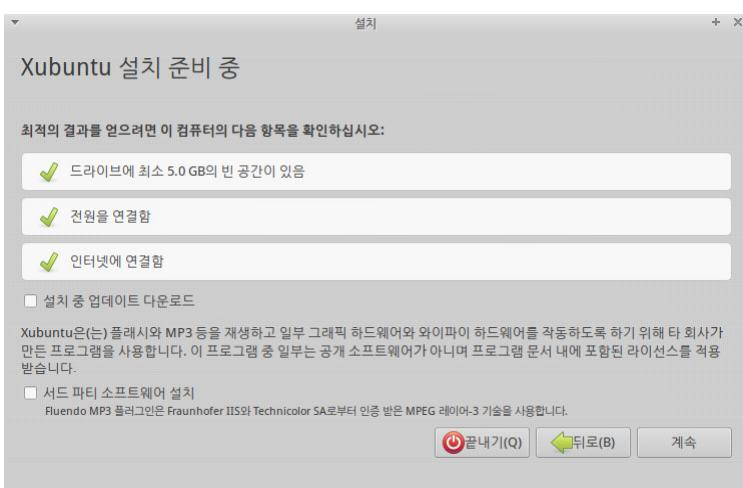
여기까지 완료하였을 경우, 본격적으로 가상 머신이 시작됩니다. 이제 호스트 운영체제에서 설정할 것은 모두 마쳤습니다. 지금부터 본격적인 리눅스 설치가 시작됩니다.

호스트 운영체제?
가상 머신에서의 용어로, 가상 머신이 돌아가는 운영체제를 호스트(host) 운영체제라 하고, 가상 머신에 설치된 운영체제를 게스트(guest) 운영체제라고 합니다.

4. 이제 호스트 운영체제에서 설정해야 하는 부분은 모두 마쳤습니다. 혹시라도 집에 쓰지 않는 노트북이 있어서 그 곳에 직접 Xubuntu를 설치하는 분들이 있다면 지금부터 따라하시면 됩니다. 이제 가상 머신을 이용해서 실제로 하나의 빈 컴퓨터에 리눅스 운영체제를 설치하는 과정을 따라가 보도록 하겠습니다.

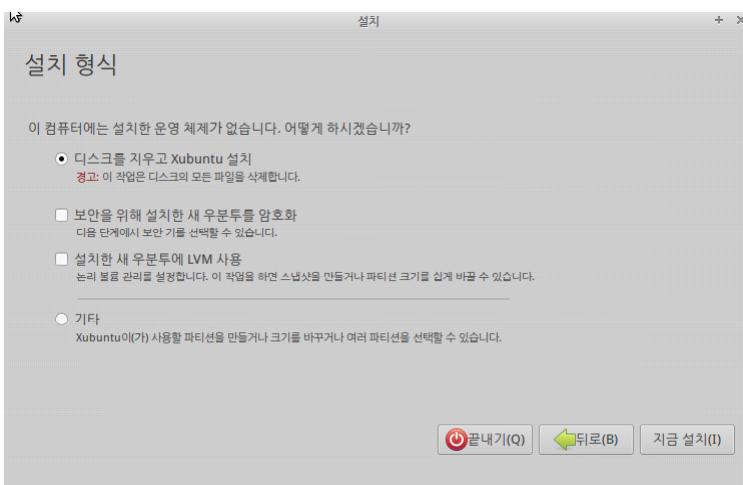


처음 Xubuntu CD를 가지고 부팅을 시도하면 다음과 같은 창이 뜹니다. CD에 있는 우분투 이미지를 가지고 우분투를 체험하는 기능까지 갖추고 있습니다. 그렇지만 우리는 가상 머신의 하드 드라이브에 우분투를 설치해야 하므로 오른쪽의 버튼을 눌러 줍시다.



설치 프로그램에서 자동으로 드라이브의 공간과 인터넷 연결 여부를 탐지합니다. Virtual PC는 처음 가상 머신을 생성할 때 기본 값으로 NAT가 잡혀 있어 호스트 운영체제의 네트워크 연결을 공유할 수 있게 됩니다. [계속]을 눌러 설치를 계속 진행합니다.

(설치 중 업데이트를 다운로드 할 수 있지만 현재 디폴트 서버가 미국으로 되어 있어 속도가 매우 느리므로 설치하지 않습니다.)



상급 유저들은 [기타]에서 홈 디렉토리나 스왑 파티션 등을 지정하기도 하지만 우리는 이제 막 리눅스를 설치하기 시작하는 유저들이므로 설치 프로그램에 모든 것을 맡깁니다. [계속]을 클릭합니다.

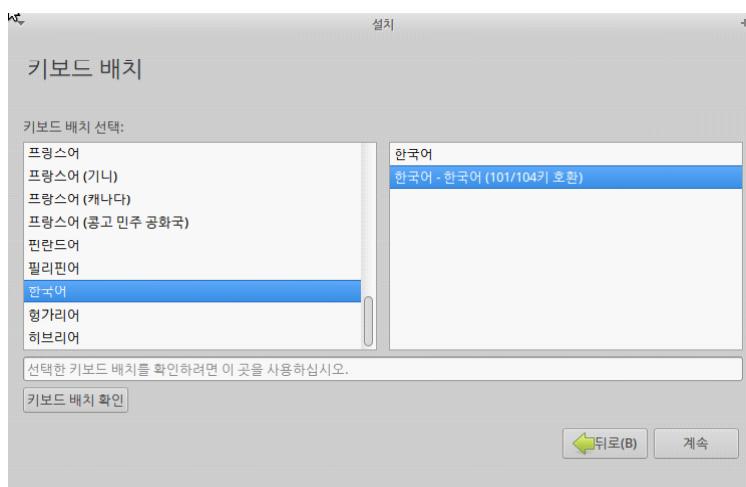
최종 확인을 하는 창이 나오면 [지금 설치] 버튼을 눌러 설치를 시작합니다.

[지금 설치(I)]

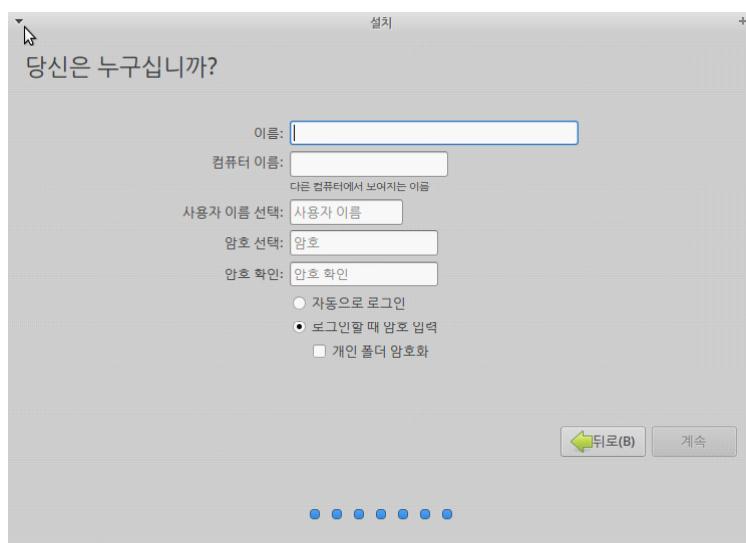
5. 설치가 시작됩니다. 설치를 진행함과 동시에 지역, 키보드, 그리고 계정에 대한 설정을 해줍니다. Windows를 설치해본 경험이 있는 분이라면 익숙한 화면입니다.



지역에 대한 설정입니다. 시간대나 언어에 대한 설정이 여기에 포함됩니다. 우리는 모두 한국 사람이므로 기본 설정값인 [Seoul]을 확인하고 계속 버튼을 눌러 줍시다.



키보드 설정을 해줍니다. 한국어를 지원하는 키보드를 사용한다는 표시를 해주는 것으로, 실제 한글 사용에 대한 것은 설치가 모두 완료된 후 다시 설정해주어야 합니다. 이에 대해서는 뒷부분에서 다시 살펴보도록 합니다.

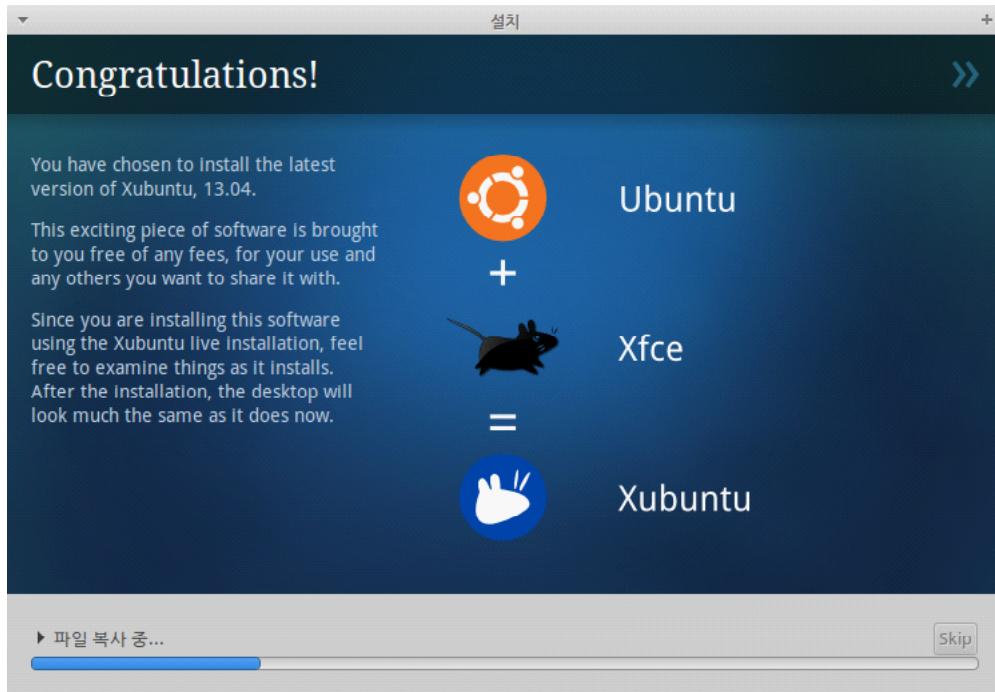


본격적으로 리눅스에서 사용할 계정을 설정해 줍니다. 리눅스로 윈도우의 Administrator와 같은 root계정이 존재하지만, 보안상의 이유로 root를 직접 사용하지는 않습니다. 사용자 이름과 암호, 그리고 로그인에 대한 설정을 해주고 계속 버튼을 눌러 줍니다.

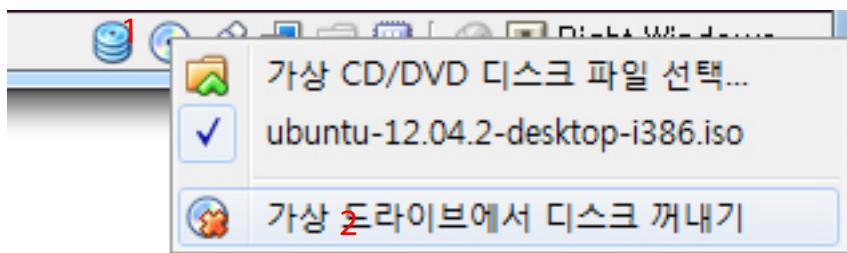
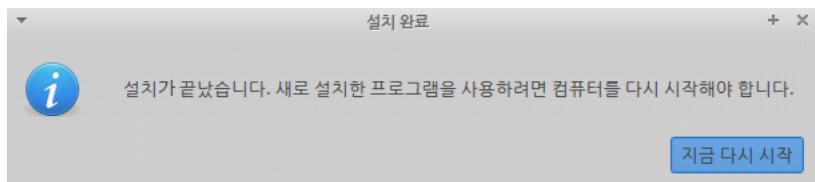
sudo?

리눅스에서는 필요할 때 root계정의 기능을 쓰거나, 콘솔을 root가 사용하는 것으로 변경할 수 있는데, 이 때 사용하는 명령어가 바로 sudo입니다. 이는 리눅스를 사용하면서 가장 많이 사용하는 명령어 중 하나입니다.

설치가 계속 진행됩니다. 가상 머신인 만큼 다른 일을 하면서 기다릴 수 있습니다. 경우에 따라 다르지만, 약 30분정도 소요됩니다.



설치가 완료되면 다음과 같은 창이 뜹니다. 다시 시작하기 전에 설치를 하는 데 쓰였던 리눅스 CD를 꺼내야 합니다. 가상 머신 하단에 있는 CD모양의 아이콘을 클릭한 뒤 디스크 꺼내기를 눌러줍니다.



이제 설치가 모두 완료되었습니다. 다음에는 리눅스 운영체제 환경을 가상 머신에서 사용하기 편리하게 하기 위한 여러가지 설정을 해보도록 하겠습니다.

Lecture

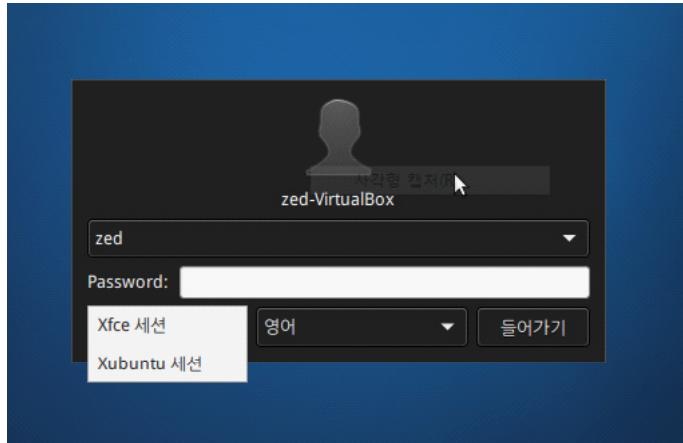
이번 강의에서는 리눅스의 기본적인 설정을 진행합니다.

설치는 크게 다음과 같은 순서로 진행됩니다.

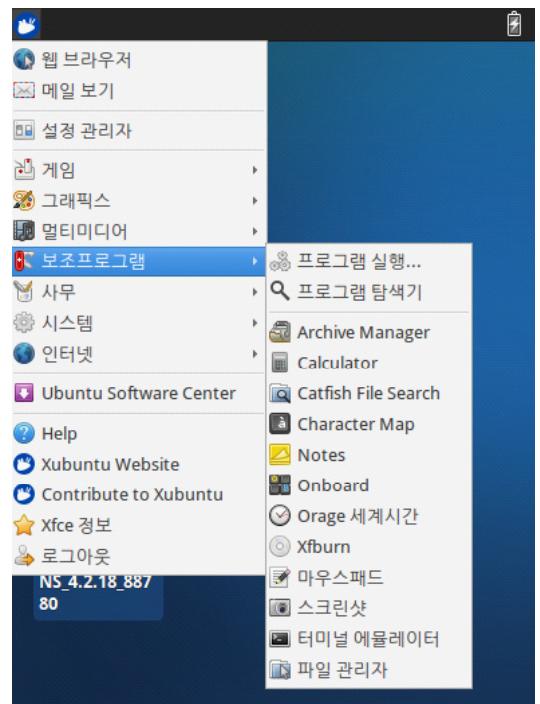
1. VirtualBox 설치
2. Xubuntu 13.04 설치
3. 소프트웨어 저장소를 다음(ftp.daum.net)으로 변경
4. 저장소 목록 업데이트 / 패키지 업그레이드
5. Guest 플러그인 설치
6. 사용언어 한글로 변경

1. Xubuntu에서는 업데이트를 하게 될 패키지들을 저장해놓은 곳을 [소프트웨어 저장소]라고 부릅니다. 업데이트를 실행할 때 바로 이 저장소에서 버전을 비교하여 상위 버전이 있을 시에 패키지 업데이트를 수행하게 됩니다. 그렇지만 기본으로 설정되어 있는 서버는 매우 느리기 때문에 우리에게 친근한 Daum에서 제공해주는 서버로 변경해 보도록 하겠습니다.

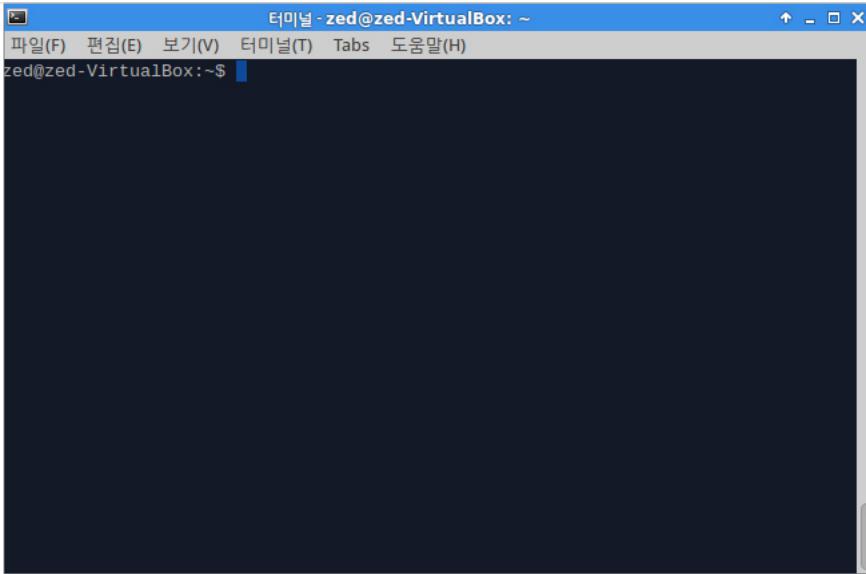
가장 먼저, 컴퓨터를 재시작 하면 로그인을 해 주어야 합니다. 특히, 세션 부분을 주목할 필요가 있는데, xfce와 Xubuntu 중 Xubuntu 세션을 선택합니다. 여기에 대한 자세한 이야기는 뒷 부분에 적도록 하겠습니다.



그 후 시작 메뉴를 이용하여 터미널을 실행시켜 줍니다.
[보조 프로그램] - [터미널 에뮬레이터]를 눌러 터미널을 실행할 수 있습니다. 또는, [시작 키 (Mac의 경우 Command) + [R]] 을 누르면 뜨는 창에서 [term]을 눌러서도 실행할 수 있습니다.



어떤 방법으로든 터미널을 실행하면, 다음과 같은 창이 뜹니다. 마치 우리가 보아오던 MS-DOS의 모습과 흡사합니다.



여기에는 다음과 같은 명령어를 입력해 줍니다.

이는 관리자 권한으로 (sudo) vi라는 프로그램을 이용하여 해당 파일을 열으라는 의미입니다.

```
$ sudo vi /etc/apt/sources.list
```

비밀번호를 입력해준 뒤에 vi화면으로 진입하게 됩니다.

[:]키를 누른 뒤에 다음 명령어를 입력합니다.

```
:%s/kr.archive.ubuntu.com/ftp.daum.net/g  
:%s/security.ubuntu.com/ftp.daum.net/g
```

리눅스 소프트웨어 저장소는 크게 두 가지로 구분되는데, 일반 소프트웨어 업데이트 저장소와, security 저장소가 그것입니다. 위 명령어는 이 두 저장소를 모두 Daum이 제공하는 서버인 [ftp.daum.net]으로 변경하라는 의미로, 추후에 vi에 대해 배우는 장에서 자세히 설명하도록 하겠습니다.

[:]를 다시 누른 뒤, wq를 눌러 해당 파일을 저장하고 빠져나옵니다.

이와 같은 과정을 마치면 업데이트 서버가 Daum으로 변경되었을 것입니다. 이를 확인하는 작업으로, 직접 업데이트와 업그레이드를 해보도록 하겠습니다.

2. Ubuntu에 설치된 패키지들을 업데이트하는 작업입니다. Update와 Upgrade라는 명령어를 이용합니다. Update는 위에서 설정한 소프트웨어 저장소와 우리의 운영체제의 패키지를 비교하여 리스트를 동기화하는 작업입니다. Upgrade는 Update된 패키지를 비교하여 갱신이 필요한 것을 다운로드 받아 설치하는 작업을 합니다. 대개 이 두 명령어는 서로 맞물려 실행됩니다.

다시 한 번, 터미널을 실행해 봅시다. 앞으로도 터미널을 많이 사용하게 될 것이므로, 바탕 화면이나 시작 메뉴에 끌어다 놓아 링크를 만들어놓는 것도 좋은 방법일 것입니다.

터미널에 다음 명령어를 입력해 봅시다.

```
$ sudo apt-get update
```

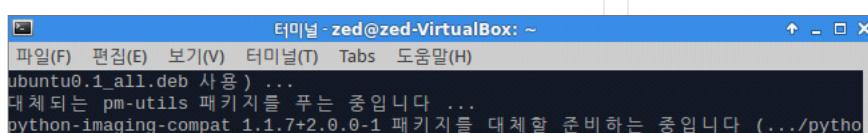
위 명령어를 입력하면, 잠깐동안 서버를 읽어들이며 동기화를 하는 화면을 볼 수 있습니다. 1분 이내로 이 작업을 완료한 이후에 아래의 명령어를 바로 입력해줍니다.

```
$ sudo apt-get upgrade
```

```
154개 업그레이드, 0개 새로 설치, 0개 제거 및 3개 업그레이드 안 함.
162 M바이트 아카이브를 받아야 합니다.
이 작업 후 20.9 M바이트의 디스크 공간을 더 사용하게 됩니다.
계속 하시겠습니까 [Y/n]? ■
```

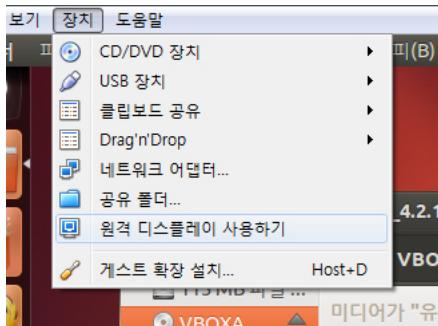
이제 update 명령을 통해 동기화된 버전 정보를 토대로 필요한 패키지를 다운로드 받아 설치하는 과정이 진행됩니다. 여러분들의 운영체제는 최초 설치 이후 많은 업데이트가 이루어졌기 때문에 설치하는 데 꽤나 오랜 시간이 걸립니다. 소프트웨어 저장소를 변경하지 않았을 시에는 훨씬 많은 시간이 소요될 수 있습니다. (여유가 있으신 분들은 비교를 해 보아도 좋습니다.)

업그레이드가 모두 완료되었을 시 터미널을 종료해 줍니다. 버튼을 눌러도 좋고, 터미널 상에서 [exit]명령을 실행시켜도 좋습니다. 실행 중인 프로세스 등이 있을 때에는 버튼을 눌러 종료가 되지 않는 경우도 있으므로 명령어를 입력하는 것을 추천합니다.



이제 여러분의 Xubuntu는 최신 상태로 구성되어 있습니다. 새롭게 어플리케이션을 깔기 위해서는 이렇게 최신 상태로 유지하는 것이 좋습니다. 그렇다면 다음에는 리눅스에서 처음으로 소프트웨어를 설치해보도록 하겠습니다.

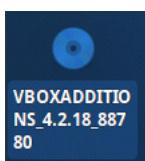
1. Guest 플러그인을 설치합니다. 이는 게스트 운영체제가 더욱 원활하게 돌아갈 수 있도록 호스트와 하드웨어, 소프트웨어적으로 연계된 다양한 기능을 제공해 주는 프로그램입니다.



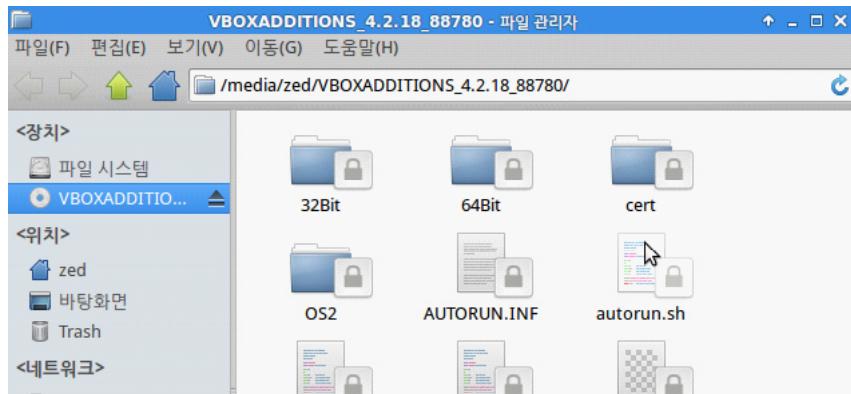
가상 머신의 장치 메뉴에서 [게스트 확장 설치]를 눌러 설치를 시작할 수 있습니다. 설치는 가상의 CD를 마운트하여 프로그램을 실행하는 방식으로 진행됩니다.

[주의사항]

이전에 Ubuntu를 설치하거나, 또는 다른 프로그램을 위해 가상 CD를 삽입한 상태라면 충돌이 발생할 수 있습니다. 따라서 본 작업을 실행하기 전에 반드시 기존의 CD를 제거해주어야 합니다.

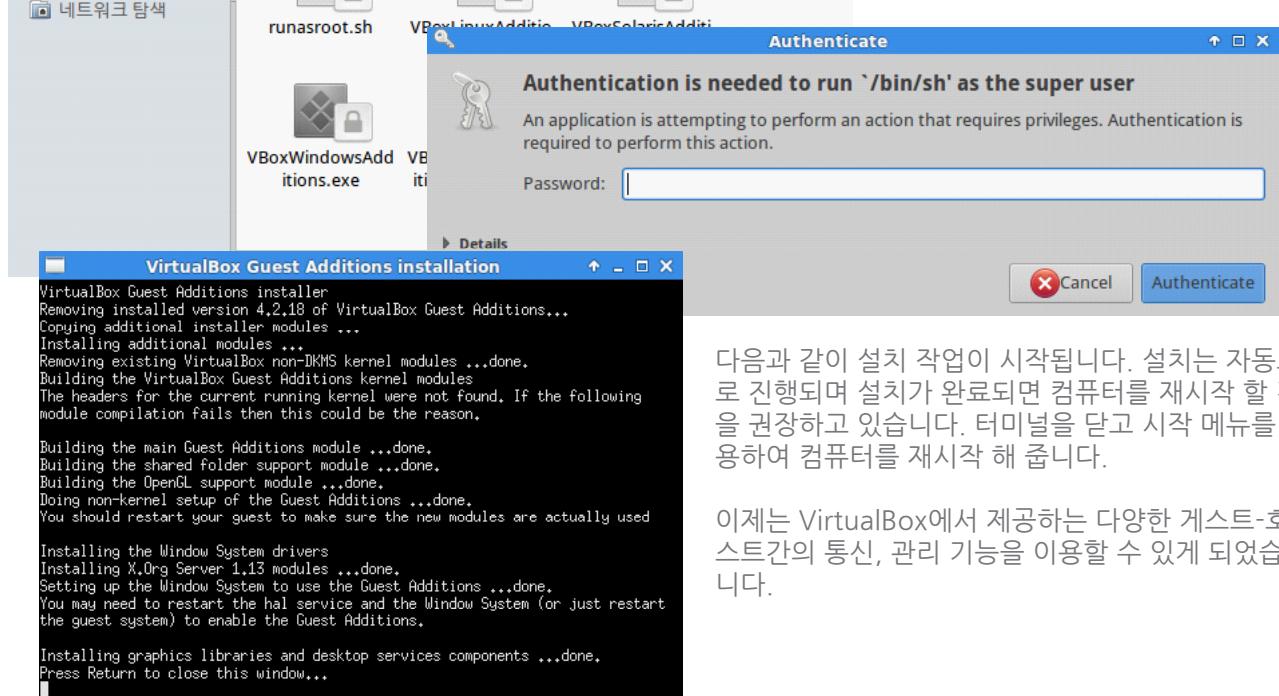


그러면 바탕화면에 다음과 같은 CD 모양의 아이콘이 생기게 됩니다. 새로운 드라이브를 마운트하였기 때문입니다. 이 버튼을 더블 클릭하여 열어줍니다.



그러면 다음과 같이 탐색기 창이 뜨게 됩니다. 여기서 autorun.sh를 실행해 줍니다. 이는 설치 과정이 자동으로 진행되도록 미리 짜여진 스크립트입니다.

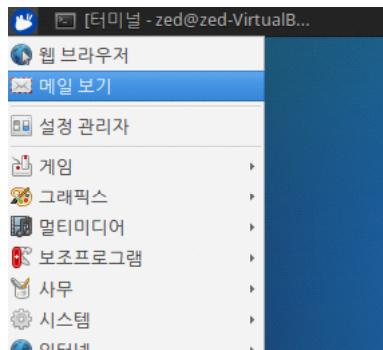
이 과정에서는 루트 권한이 필요하기 때문에 지금 사용하고 있는 계정의 비밀번호를 입력하여 루트 기능을 수행할 수 있도록 합니다.



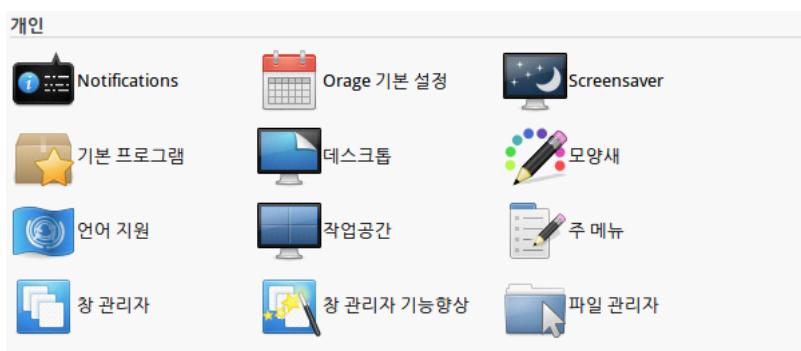
다음과 같이 설치 작업이 시작됩니다. 설치는 자동으로 진행되며 설치가 완료되면 컴퓨터를 재시작 할 것을 권장하고 있습니다. 터미널을 닫고 시작 메뉴를 활용하여 컴퓨터를 재시작 해 줍니다.

이제는 VirtualBox에서 제공하는 다양한 게스트-호스트간의 통신, 관리 기능을 이용할 수 있게 되었습니다.

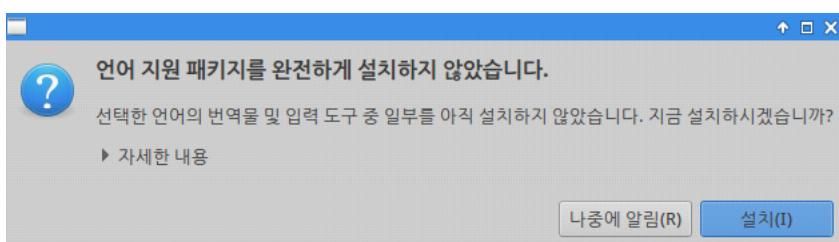
4. 마지막 초기 설정은 언어에 관한 것입니다. Xubuntu는 설치 직후 완전한 한글 환경을 제공하지 않습니다. 따라서 이번 강의의 마지막으로 한/영 키 전환과 한글 인터페이스에 대한 설정을 해보도록 하겠습니다.



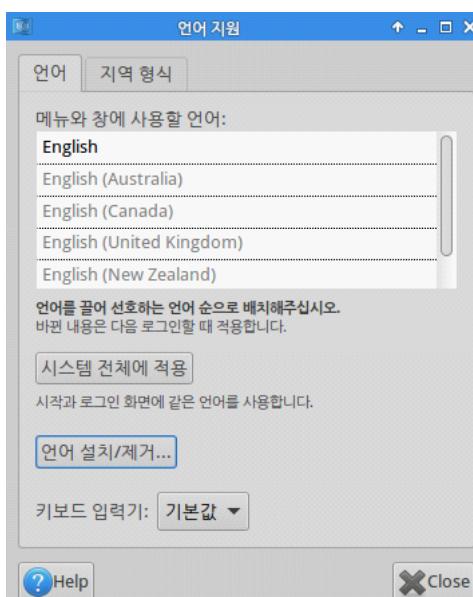
시작 메뉴를 누르고 [설정 관리자] 버튼을 눌러줍니다. 이는 Windows의 [제어판]과 동일한 역할을 하며, 각 영역별로 다양한 시스템 설정을 할 수 있습니다.



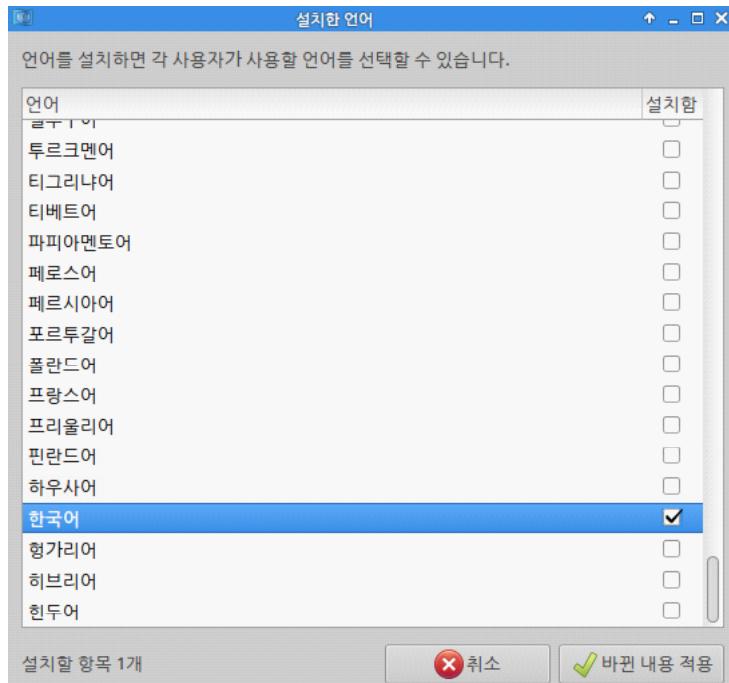
설정 관리자 창이 열리면 [언어 지원]을 눌러줍니다. 이는 언어에 관한 표시 형식이나 번역 등 인터페이스 설정을 지원하는 제어판 메뉴입니다.



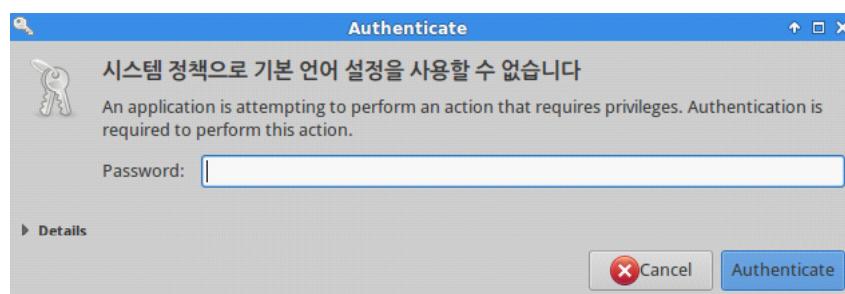
최초 실행 시 다음과 같은 창이 뜹니다. [설치]를 눌러서 언어 지원 패키지를 설치한 후에 언어 지원 설정 창으로 진입할 수 있게 됩니다.



기본 언어가 English로 세팅되어 있음을 확인할 수 있습니다. 여기에서 [언어 설치/제거] 버튼을 눌러줍니다.



다음과 같은 창이 뜨면 아래에 있는 [한국어]를 체크하여 설치하여 줍니다. [바뀐 내용 적용]을 누르면 시스템이 자동으로 언어를 설치해 줍니다.



이 작업 역시 소프트웨어 설치와 같이 시스템을 변경하는 작업이기 때문에 루트 권한을 요구합니다. 비밀번호를 입력하고 [Authenticate]버튼을 눌러 줍니다.



그러면 이후에 시스템이 자동으로 언어 설치를 마치게 됩니다. 13.04부터는 자동으로 최종 설치한 언어를 기본 언어로 설정해 주게 됩니다. [시스템 전체에 적용]을 눌러줍니다. 옆 탭의 [지역 형식] 역시 자동으로 한국어로 변경되게 됩니다. 이제 시스템을 재시작하면 변경한 언어 설정이 적용되어 있을 것입니다.

이제, 리눅스를 활용할 준비가 모두 끝났습니다.

Section 2.

Linux Server Basic

이번 강의에서는 리눅스의 프로그램들 중 설치가 가장 쉬운 것들 중 하나인 서버 모듈 APM과 콘솔 접속을 위한 서버-클라이언트 프로그램인 SSH를 설치합니다.

Supplies

SSH

SSH는 콘솔에서 네트워크를 통해 다른 시스템의 콘솔로 접속하여 컨트롤할 수 있도록 해주는 도구입니다. Mac과 Linux에서는 모두 이 SSH를 설치하여 (또는 기본 지원하여) 사용할 수 있습니다. Windows에서는 Putty라는 프로그램을 대체로 사용합니다.



APM(Apache + PHP + MySQL)

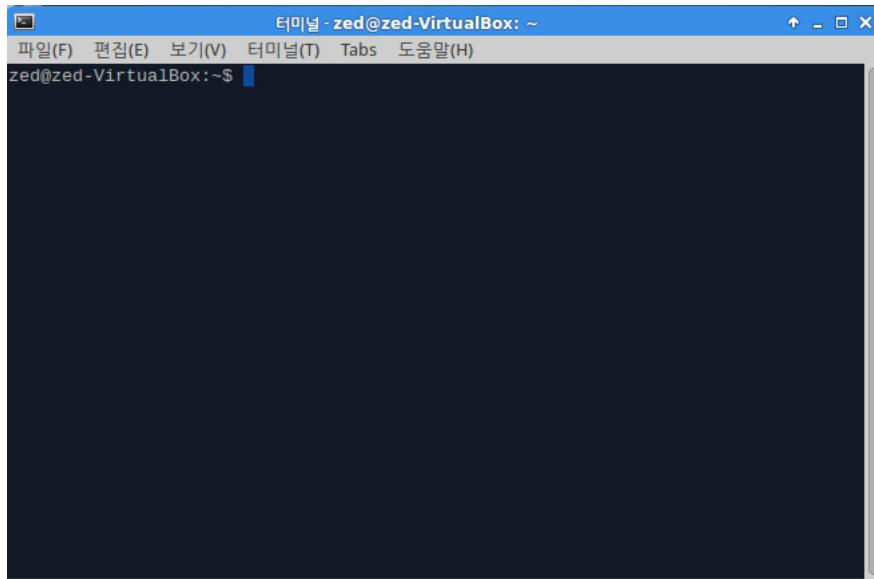
APM은 Windows와 Linux에서 모두 많은 사용자를 거닐고 있는 기본 웹 서버 set입니다. 각각 웹 서버를 여는 Apache와, 스크립트 언어를 지원하는 PHP, 그리고 무료 라이센스의 데이터베이스 프로그램인 MySQL로 이루어져 있습니다.

Lecture

이번에는 호스트에서 콘솔을 통해 게스트 콘솔을 이용할 수 있게 할 수 있도록 SSH라는 프로그램을 설치합니다.

- 터미널을 실행합니다. 터미널을 실행하는 방법에는 시작 메뉴에서 찾는 방법과, [시작 키 + R]을 누르고 term을 입력하는 방법이 있습니다.

다음과 같이 터미널 창이 실행된 것을 확인할 수 있습니다.

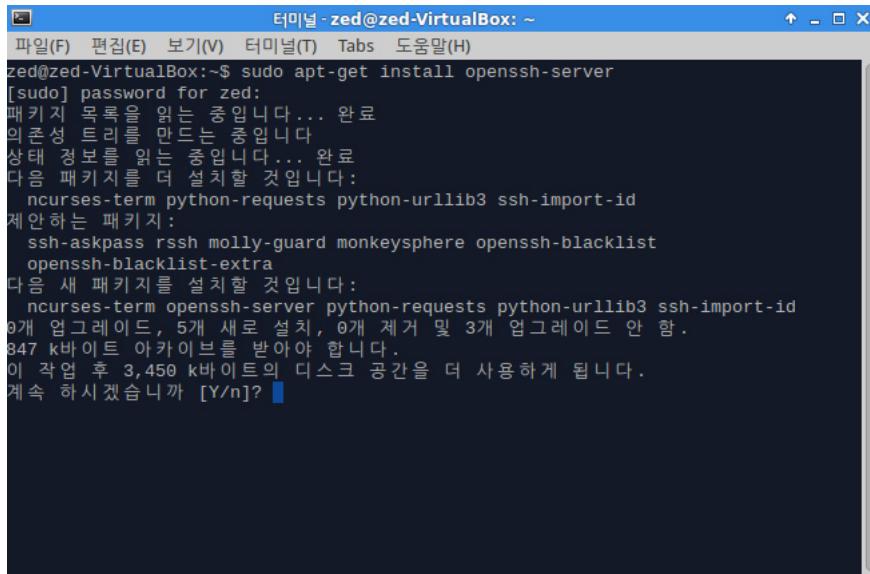


여기서 다음과 같은 명령어를 입력해 줍니다.

```
$ sudo apt-get install openssh-server
```

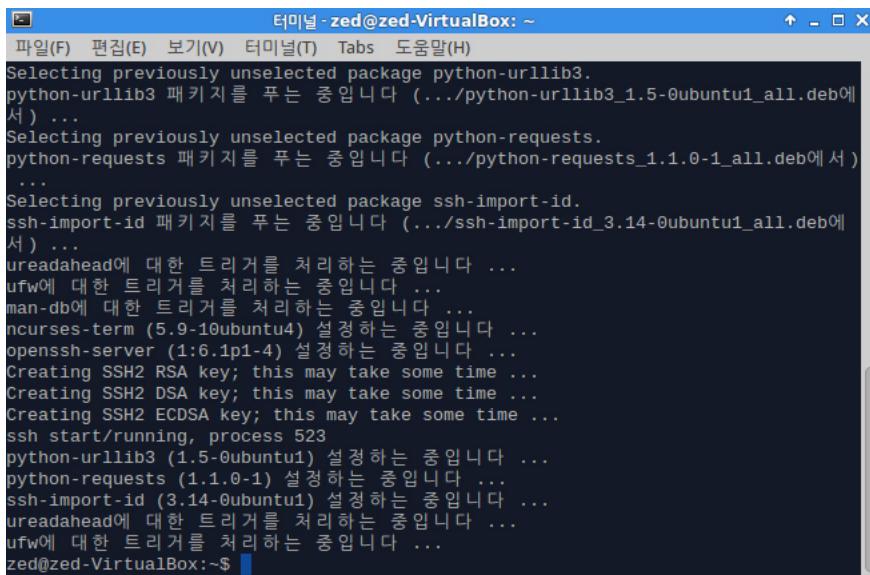
이 명령어는 sudo(관리자 권한으로) + apt-get(패키지 관리를 하는데) + install(설치를 한다.) + openssh-server(라는 프로그램을)의 뜻을 가지고 있습니다. 일전에 apt-get update와 upgrade를 사용해본 적이 있으므로 익숙할 것입니다. 프로그램을 설치하기 전에 upgrade 작업을 하여 동기화시키고, 설치 후에 update 작업을 통해 다른 프로그램들을 동기화하고, 현재 작업을 확인하는 것도 좋은 방법입니다.

명령어를 입력하면 계정의 비밀번호를 입력하라는 메시지가 뜹니다. 이는 패키지 설치가 루트 권한을 요구하기 때문으로, 이미 게스트 프로그램을 설치할 때 경험한 적이 있습니다. 암호를 입력해주면 다음과 같이 설치를 확인하는 문구가 뜹니다. [Y]를 누르거나 그냥 엔터를 누르면 동의를 한다는 표시가 됩니다.



```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
zed@zed-VirtualBox:~$ sudo apt-get install openssh-server
[sudo] password for zed:
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
다음 패키지를 더 설치할 것입니다:
 ncurses-term python-requests python-urllib3 ssh-import-id
제안하는 패키지:
 ssh-askpass rssh molly-guard monkeysphere openssh-blacklist
 openssh-blacklist-extra
다음 새 패키지를 설치할 것입니다:
 ncurses-term openssh-server python-requests python-urllib3 ssh-import-id
0개 업그레이드, 5개 새로 설치, 0개 제거 및 3개 업그레이드 안 함.
847 kB 이트 아카이브를 받아야 합니다.
이 작업 후 3,450 kB 이트의 디스크 공간을 더 사용하게 됩니다.
계속 하시겠습니까 [Y/n]? 
```

패키지 설치에 동의를 하면 시스템에서 패키지와 패키지를 실행하기 위해 필요한 다른 패키지들을 모두 설치해 줍니다. 설치를 마치면 다시 입력이 가능한 프롬프트 형태로 돌아오게 됩니다.



```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
Selecting previously unselected package python-urllib3.
python-urllib3 패키지를 푸는 중입니다 (.../python-urllib3_1.5-0ubuntu1_all.deb에서) ...
Selecting previously unselected package python-requests.
python-requests 패키지를 푸는 중입니다 (.../python-requests_1.1.0-1_all.deb에서) ...
...
Selecting previously unselected package ssh-import-id.
ssh-import-id 패키지를 푸는 중입니다 (.../ssh-import-id_3.14-0ubuntu1_all.deb에서) ...
ureadahead에 대한 트리거를 처리하는 중입니다 ...
ufw에 대한 트리거를 처리하는 중입니다 ...
man-db에 대한 트리거를 처리하는 중입니다 ...
ncurses-term (5.9-10ubuntu4) 설정하는 중입니다 ...
openssh-server (1:6.1p1-4) 설정하는 중입니다 ...
Creating SSH2 RSA key; this may take some time ...
Creating SSH2 DSA key; this may take some time ...
Creating SSH2 ECDSA key; this may take some time ...
ssh start/running, process 523
python-urllib3 (1.5-0ubuntu1) 설정하는 중입니다 ...
python-requests (1.1.0-1) 설정하는 중입니다 ...
ssh-import-id (3.14-0ubuntu1) 설정하는 중입니다 ...
ureadahead에 대한 트리거를 처리하는 중입니다 ...
ufw에 대한 트리거를 처리하는 중입니다 ...
zed@zed-VirtualBox:~$ 
```

이제 게스트 시스템에서 ssh를 사용하기 위한 과정은 모두 마쳤습니다. 다음에는 호스트 시스템에서 역시 ssh를 이용하여 게스트 시스템의 콘솔로 연결을 시도해 보도록 하겠습니다.

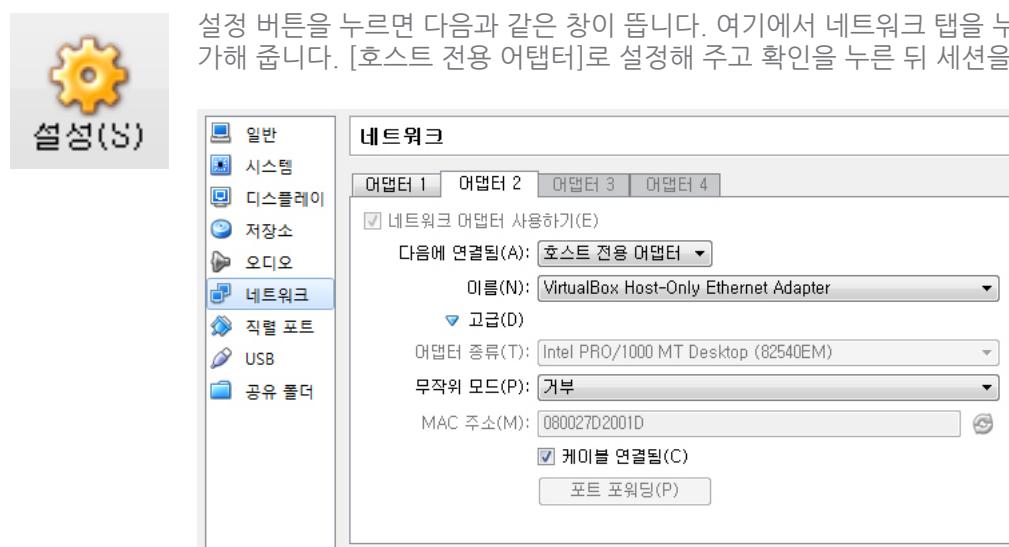
2. 호스트 시스템에서 게스트 시스템의 콘솔로 들어가 봅니다. 이 교재에서는 Windows를 기준으로 설명합니다. 수업에서는 아마 Mac을 기준으로 설명할 것입니다. 그렇지만 접속 프로그램의 차이에 불과하므로 Mac에 대해서도 간략하게 정리를 해 두도록 하겠습니다.

가장 먼저 putty를 설치합니다. Windows에는 ssh가 없기 때문에 이 프로그램을 이용해서 ssh 서버에 접속할 수 있습니다. 이 프로그램은 <http://www.chiark.greenend.org.uk/~sgtatham/putty/>에서 받을 수 있습니다.

For Windows on Intel x86

PUTTY:	putty.exe	(or by FTP)	(RSA sig)	(DSA sig)
PUTTYtel:	puttytel.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSCP:	pscp.exe	(or by FTP)	(RSA sig)	(DSA sig)
PSFTP:	psftp.exe	(or by FTP)	(RSA sig)	(DSA sig)
Plink:	plink.exe	(or by FTP)	(RSA sig)	(DSA sig)
Pageant:	pageant.exe	(or by FTP)	(RSA sig)	(DSA sig)
PUTTYgen:	puttygen.exe	(or by FTP)	(RSA sig)	(DSA sig)

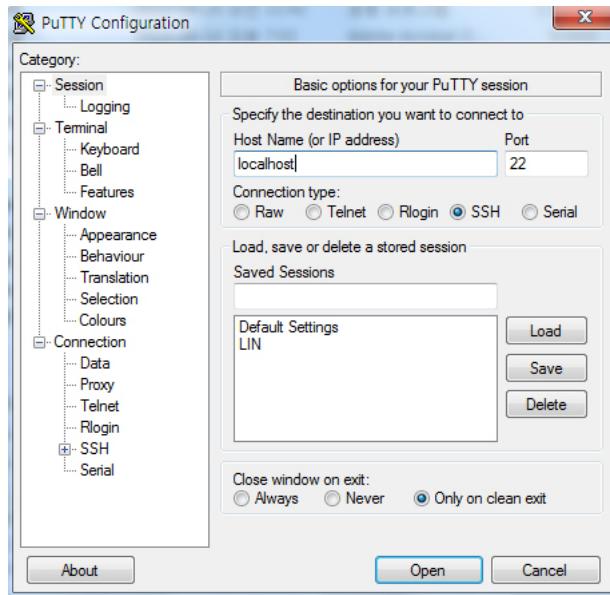
가장 위에 있는 putty.exe를 받으면 따로 설치하지 않고도 putty를 이용할 수 있게 됩니다. 그렇지만 프로그램을 실행하기 전에 게스트와 호스트를 이어주는 설정이 필요합니다. 잠시 게스트 운영체제를 종료하고 설정 버튼을 눌러 줍니다.



다음으로 다시 터미널을 열고 ifconfig이라는 명령어를 입력해 줍니다. eth0, eth1, lo라는 세 개의 정보가 들 것입니다. 순서대로 어댑터 1, 어댑터 2, 그리고 로컬 전체의 외부 IP주소입니다. 우리가 필요한 것은 eth1의 IP주소이므로 이를 기억해둡니다.

```
eth1      Link encap:Ethernet HWaddr 08:00:27:d2:00:1d
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed2:1d/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:126 errors:0 dropped:0 overruns:0 frame:0
            TX packets:35 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:14368 (14.3 KB)  TX bytes:6571 (6.5 KB)
```

이제 putty를 실행해 줍니다. IP주소를 쳐야 하는데, 컴퓨터에 대해 어느정도 아는 경우 여기에 로컬이므로 local-host 또는 127.0.0.1을 치는 경우가 생기는데, 아까 말했다시피 서로간 IP주소를 공유하는 것이 아니기 때문에 이는 먹히지 않습니다. 아까 알아두었던 eth1의 어댑터 아이피 주소를 입력해 주고 접속 버튼을 눌러줍니다.



로그인을 하라는 메시지가 뜹니다. 아이디와 패스워드를 입력하고 서버에 접속하면 다음과 같이 터미널 화면과 같은 콘솔이 나타난 것을 확인할 수 있습니다.

```
zed@zed-VirtualBox: ~
login as: zed
zed@192.168.56.102's password:

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Welcome to Ubuntu 13.04 (GNU/Linux 3.8.0-19-generic i686)

 * Documentation:  https://help.ubuntu.com/

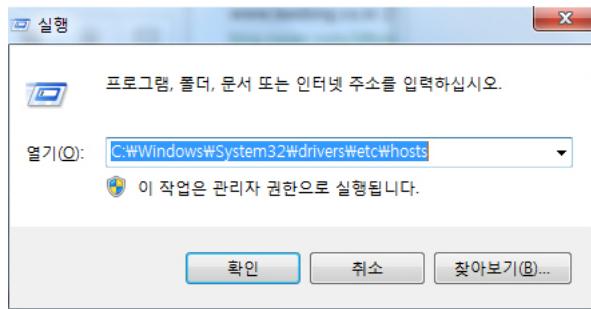
7 packages can be updated.
7 updates are security updates.

zed@zed-VirtualBox:~$
```

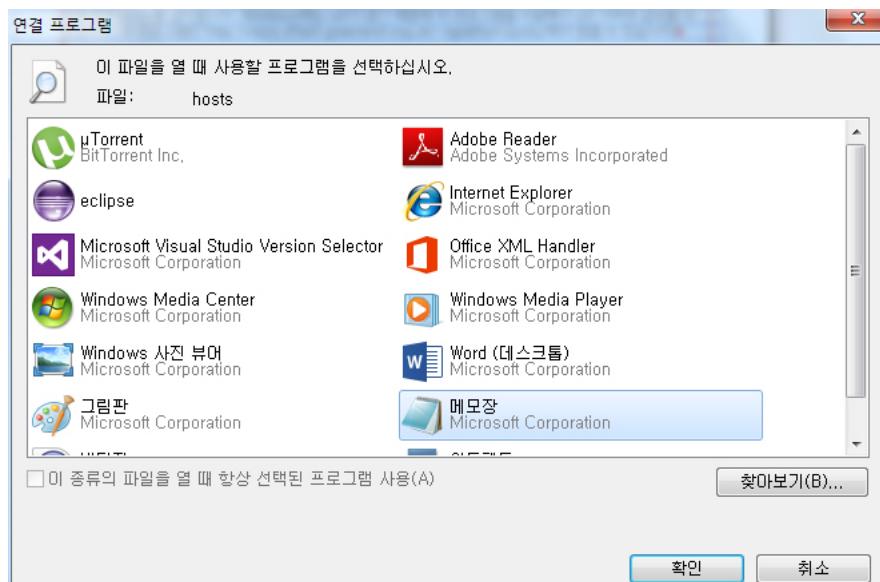
그런데 IP가 매우 보기 좋지 않습니다. 다음에는 내부에서만 사용할 수 있는 IP에 일정한 도메인을 매치시키는 방법을 이용해서 자기만의 도메인처럼 꾸며보도록 하겠습니다.

3. hosts를 이용하여 도메인을 설정해봅시다. 이 파일에 아이피 주소와 매치하고자 하는 도메인 주소를 입력하면 마치 도메인을 구매한 것처럼 인식하여 해당 아이피에 문자를 이용하여 접속할 수 있게 됩니다.

호스트 시스템에서 [시작 버튼 + R]을 열어 [실행] 대화 상자를 실행시켜 줍니다. 그 곳에서 C:\Windows\System32\drivers\etc\hosts를 실행시켜 줍니다. 이 파일은 관리자만 수정할 수 있는데, 실행 대화 상자에서 여는 경우 자동으로 관리자 권한으로 실행되게 됩니다.



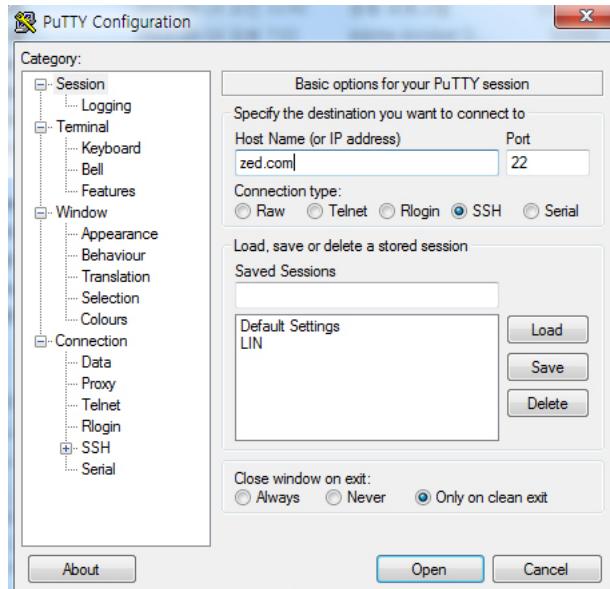
파일 확장자가 존재하지 않으므로 [연결 프로그램] 대화 상자가 뜹니다. 인식 가능한 텍스트로 이루어져 있으므로 [메모장]이나 [워드패드]를 가지고 편집할 수 있습니다.



텍스트 편집기가 뜨면 가장 하단에 알아두었던 어댑터의 ip주소와 원하는 주소를 입력해 줍니다. [Tab]키를 이용하면 쉽게 단에 맞게 구분이 가능해 집니다. 입력을 완료하였으면 저장 버튼을 누르고 프로그램을 종료합니다.

```
# localhost name resolution is handled within DNS itself.
#       127.0.0.1      localhost
#       ::1      localhost
192.168.56.102 zed.com
```

다시 putty를 실행하여 입력하였던 도메인 네임을 이용하여 접속을 시도하여 봅시다.



아까와 같은 상태로 접속이 원활하게 이루어질 경우 설정이 제대로 된 것입니다. 이제 이것을 이용하여 게스트에서 웹 서버를 구축하고 수정하는 것을 호스트 브라우저를 이용하여 확인이 가능해 집니다. 그렇다면 이제부터 본격적으로 간단한 웹 서버를 구축해 보도록 하겠습니다.

Advanced : Mac 환경에서 SSH 구축하기

Mac에는 기본적으로 ssh가 내장되어 있기 때문에 다음과 같은 명령어를 터미널에 입력하여 접속할 수 있다.

```
$ ssh <id>@<host address>
```

Windows 환경과 마찬가지로 hosts 파일을 수정함으로써 개인 가상 도메인 설정 역시 가능하다.
맥 터미널에서 /etc/hosts 파일에 아래 줄을 추가해주면 된다.

```
<ip> my_domain.net
```

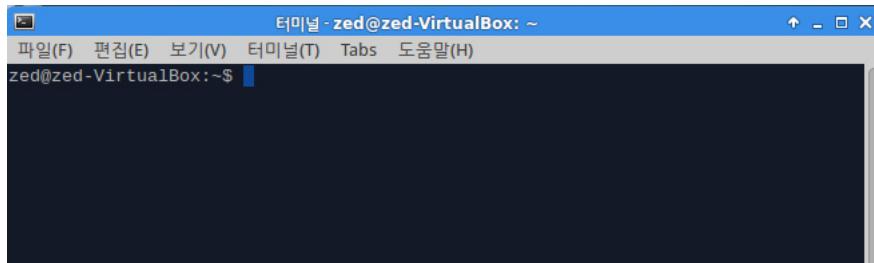
이 작업을 마치면 호스트에서 도메인을 이용해 접속이 가능해지게 된다.

Lecture

이번에는 Apache를 통해 리눅스의 간단한 웹 서버를 구축하고, 거기에 PHP와 MySQL을 연결하여 기본적인 기능을 하는 서버를 완성시키는 작업을 합니다.

- 역시 가장 먼저 프로그램을 실행해야 하므로, 터미널을 실행합니다. 터미널은 앞으로도 계속해서 사용되므로 바탕 화면이나 메뉴에 링크를 만들어 두는 것이 좋습니다.

다음과 같이 터미널 창이 실행된 것을 확인할 수 있습니다.

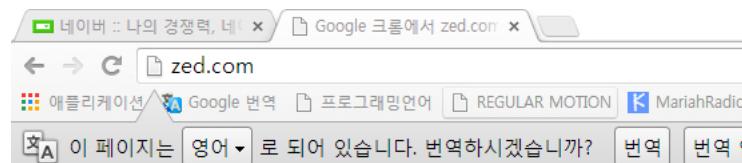


가장 먼저 설치해야 할 모듈은 웹 서버인 apache2입니다. 이것을 설치하면 브라우저에서 html이나 txt, 그림 파일 등의 기본적인 파일을 읽어들일 수 있게 됩니다. 터미널에서 다음과 같은 명령어를 입력해 줍니다.

```
$ sudo apt-get install apache2
```

install 이후에 [Tab]키를 이용할 경우 더욱 쉽게 프로그램을 찾을 수 있습니다. 위와 같은 경우 apache까지만 치고 [Tab]키를 누를 경우 2라는 숫자가 자동으로 입력됨을 확인할 수 있습니다. 역시 비밀번호를 입력하여 프로그램을 설치합니다.

이 작업 이후에 호스트 시스템에서 이전에 정하였던 가상 도메인을 가지고 접속해 보면, 이와 같은 창이 뜨는 것을 확인할 수 있습니다.



It works!

This is the default web page for this server.

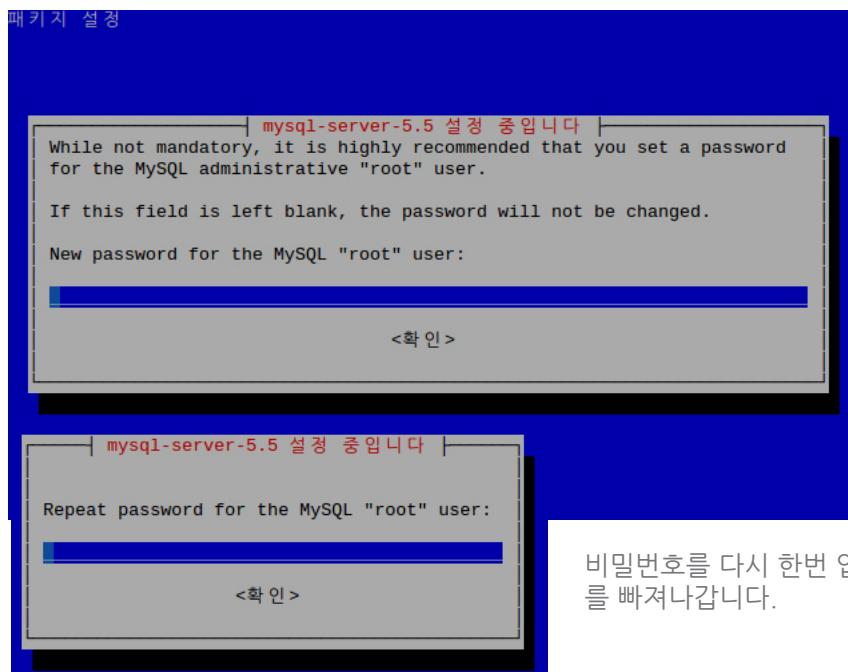
The web server software is running but no content has been added, yet.

2. 이제 php와 MySQL을 설치합니다. php는 html에 포함되는 스크립트 언어로 널리 사용되고 있으며, 보통 데이터베이스와 함께 맞물려 동작합니다. MySQL은 강력한 무료 라이센스 DBMS로, 이 두 가지가 있으면 기본적인 기능을 모두 수행할 수 있는 웹 서버가 완성됩니다.

역시 터미널에서 설치를 하는 것부터 시작합니다. 가장 먼저 MySQL을 다음 명령어를 입력함으로써 설치해 줍니다.

```
$ sudo apt-get install mysql-server
```

이전과 같은 패키지 설치 과정이 이루어집니다. 설치가 모두 완료될 때 쯤 root 암호를 설정하는 대화 상자가 열리게 됩니다. 이 비밀번호를 잊어버릴 시에는 복구가 매우 복잡하기 때문에 잊어버리지 않도록 주의하시길 바랍니다.



비밀번호를 다시 한번 입력하고 [Enter] 키를 눌러 대화 상자를 빠져나갑니다.

이 작업을 마치면 MySQL의 설치는 마친 것입니다. 그 다음으로 아래 명령어를 입력하여 php를 설치해 줍니다. php는 따로 설정이 필요 없이 바로 설치가 될 것입니다.

```
$ sudo apt-get install php5
```

이제 이것을 토대로 미리 수업에서 제공하는 템플릿을 가지고 서버가 올바르게 동작하는지 체크해보고, 간단한 실습을 함께 해보도록 하겠습니다.

3. 지금까지 설치한 웹 서버가 실제로 잘 돌아가는지를 확인해 봅시다. 실습은 미리 준비된 템플릿을 받아서 웹 서버에 옮겨다 놓는 방식으로 진행됩니다.

가장 먼저 홈 디렉토리로 이동합니다. 홈 디렉토리로 이동하는 방법은 터미널에 [cd ~], 또는 [cd #]를 입력하면 됩니다. 이는 자신의 홈 디렉토리에 접근하는 가장 빠른 방법들이니 기억해 두시길 바랍니다.

홈 디렉토리에서 인터넷 저장소에 있는 파일을 받아올텐데, 우리는 git이라는 도구를 사용합니다. 지금 이 책을 보시는 분들은 대부분 이전 학기에 git을 다룬 적이 있으실 것입니다. 그렇지만 Xubuntu에는 기본적으로 git이 설치되어있지 않으므로 다음 명령어를 이용하여 git을 설치하여 줍니다.

```
$ sudo apt-get install git
```

git이 설치되고 나서는, 저장소를 복사해 주어야 합니다. 저장소 복사는 clone이라는 명령을 가지고 할 수 있는데, 아래의 저장소를 그대로 복사해 옵니다.

```
$ git clone https://github.com/horong54/133linux.git
```

복사된 저장소의 week1 디렉토리로 들어가보면 다음과 같이 미리 작성된 데이터베이스 파일과 php파일 등이 있는 것을 확인할 수 있습니다. 우리는 이 파일들을 미리 설정된 웹 서버 디렉토리로 옮겨주어야 합니다.

The screenshot shows a terminal window titled '터미널 - zed@zed-VirtualBox: ~/133linux/week1'. It displays the command 'git clone https://github.com/horong54/133linux.git' being run, followed by the output of the cloning process. The output includes messages about selecting packages, compressing objects, and unpacking objects. After cloning, the user navigates to the 'week1' directory and lists its contents, which include several PHP files and SQL scripts.

```

터미널 - zed@zed-VirtualBox: ~/133linux/week1
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
Selecting previously unselected package git-man.
git-man 패키지를 푸는 중입니다 (.../git-man_1%3a1.8.1.2-1_all.deb에서) ...
Selecting previously unselected package git.
git 패키지를 푸는 중입니다 (.../git_1%3a1.8.1.2-1_i386.deb에서) ...
man-db에 대한 트리거를 처리하는 중입니다 ...
liberror-perl (0.17-1) 설정하는 중입니다 ...
git-man (1:1.8.1.2-1) 설정하는 중입니다 ...
git (1:1.8.1.2-1) 설정하는 중입니다 ...
zed@zed-VirtualBox:/home$ #
zed@zed-VirtualBox:/home$ cd #
zed@zed-VirtualBox:~$ git clone https://github.com/horong54/133linux.git
Cloning into '133linux'...
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 20 (delta 4), reused 15 (delta 2)
Unpacking objects: 100% (20/20), done.
zed@zed-VirtualBox:~$ cd 133linux
zed@zed-VirtualBox:~/133linux$ ls
README.md  week1
zed@zed-VirtualBox:~/133linux$ cd week1
zed@zed-VirtualBox:~/133linux/week1$ ls
input.html  mtable.py  php_test.php  query2.sql  user.sql
insert.php  mysql_test.php  query1.sql  readme.txt
zed@zed-VirtualBox:~/133linux/week1$ 
```

아파치 메인 계정의 디렉토리는 /var/www인데, 여기에 php 파일들을 모두 옮겨 브라우저에서 실행해 봅시다. html이 아닌 php파일이 정상적으로 실행된다면 php가 올바르게 설치된 것입니다. 다음 명령어를 입력하여 php파일만 다른 디렉토리로 옮길 수 있습니다.

```
$ sudo cp 133linux/week1/*.php /var/www
```

위 명령어는 관리자 권한으로 133linux/week1에 php로 끝나는 모든 파일들을 /var/www로 옮기라는 의미를 가지고 있습니다. 명령어를 실행해보고 해당 디렉토리에 가서 확인하면 php파일들이 모두 옮겨진 것을 확인할 수 있습니다.

이제 호스트 시스템의 웹 브라우저를 이용하여 `php_test.php` 파일을 실행해 봅시다.

System	Linux zed-VirtualBox 3.8.0-19-generic #30-Ubuntu SMP Wed May 1 16:36:13 UTC 2013 i686
Build Date	Jul 15 2013 18:03:31
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/10-pdo.ini
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525
Zend Extension Build	API20100525,NTS
PHP Extension Build	API20100525,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled

위와 같은 페이지가 정상적으로 실행되면 php가 올바르게 설치된 것입니다. 다음으로는 MySQL의 설치를 확인해야 하는데, 그러면서 데이터베이스에 테이블을 생성하고 값을 집어넣는 간단한 작업을 함께 해보도록 하겠습니다. 다시 콘솔로 돌아가서 작업을 시작합니다. 여기서의 콘솔은 ssh로 연결된 게스트 시스템의 콘솔이어도 상관이 없습니다.

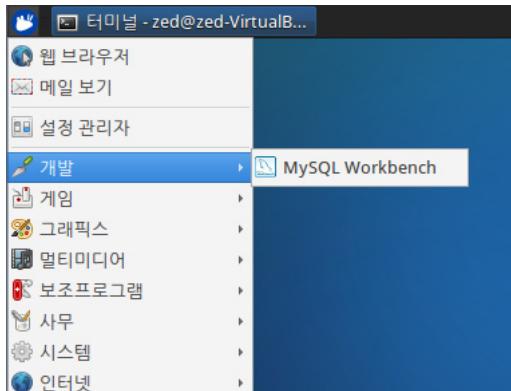
먼저 초기 설정을 해야하는데, 초기에는 이 작업을 하기 위해서는 콘솔에서 MySQL을 실행시켜 다음과 같은 명령어를 순서대로 입력해 주어야 했습니다.

```
$ mysql -u root -p
$ mysql> create database next_db;
$ mysql> create user 'next'@'localhost' identified by 'next0708';
$ mysql> GRANT ALL PRIVILEGES ON next_db.* TO 'next'@'localhost' WITH GRANT OPTION;
$ mysql> quit
```

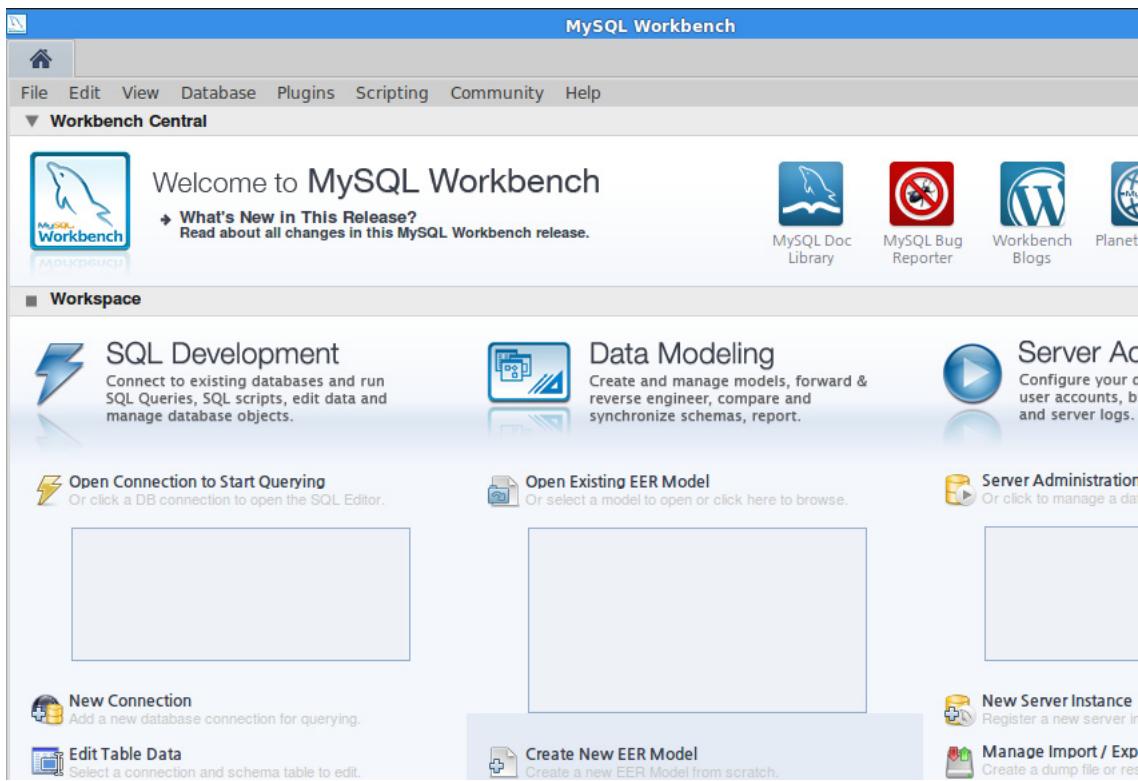
그러나 최근에는 workbench 강력하고 편리한 MySQL 관리 툴을 제공하여 콘솔에서 작업을 하지 않아도 쉽게 데이터베이스를 직관적으로 다룰 수 있게 되었습니다. 우리는 이 workbench를 이용하여 데이터베이스를 다루어 보도록 하겠습니다. 다음 명령어도 workbench를 설치합니다.

```
$ sudo apt-get install mysql-workbench
```

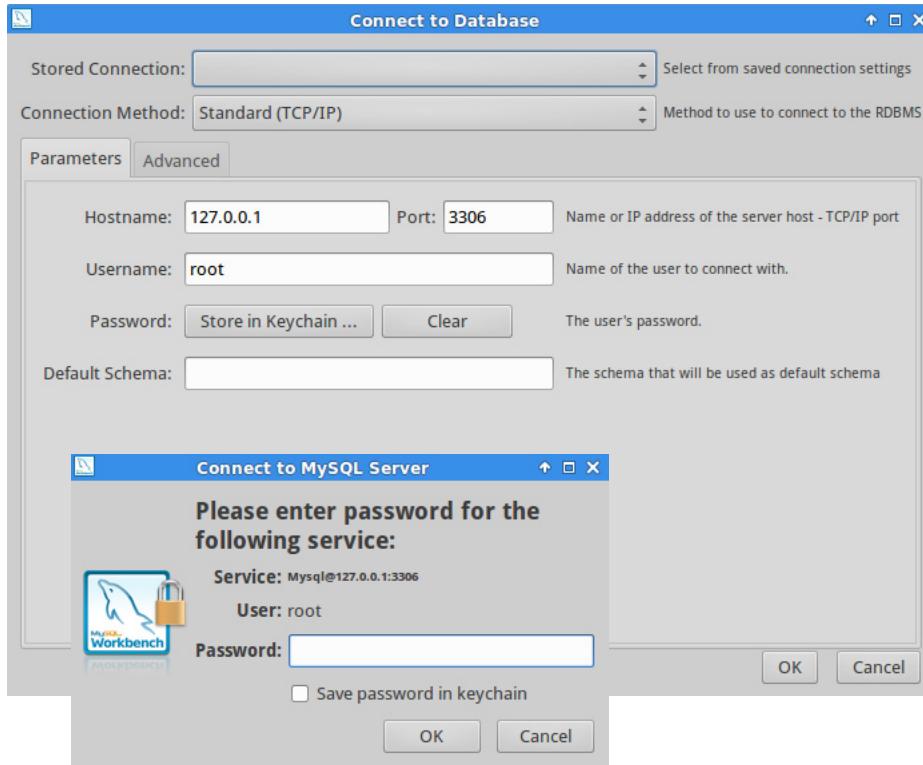
이제 시작 메뉴를 확인해보면 [개발]탭에 MySQL Workbench가 추가된 것을 확인할 수 있습니다. 이것을 클릭하여 실행시켜 봅시다.



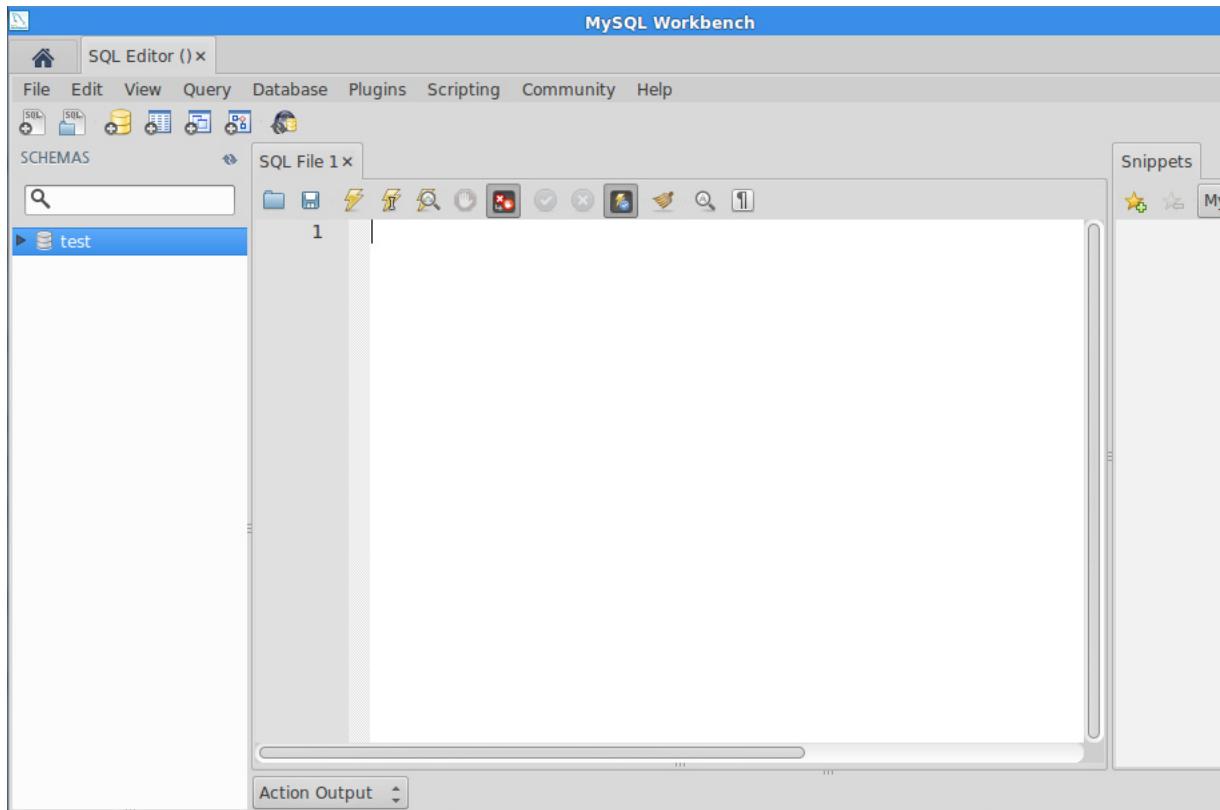
다음과 같은 창이 뜨는 것을 확인할 수 있습니다. 중간 부분에 있는 [Open Connection To Start Querying] 버튼을 눌러서 서버의 데이터베이스와 연결을 시도합니다.



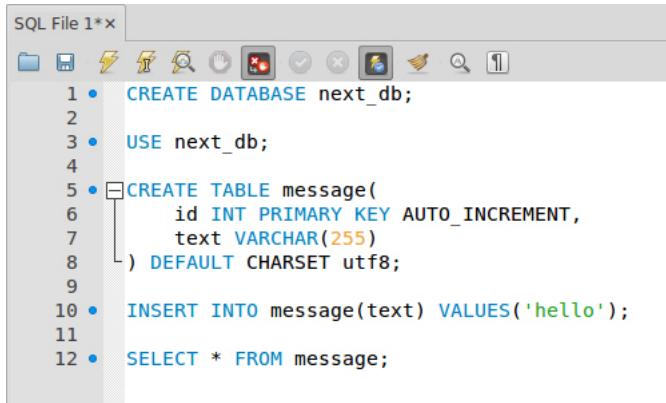
아래와 같은 창이 뜨면 MySQL을 설치할 때 입력했던 루트 비밀번호를 입력해 줍니다. 암호를 입력할 때 아래에 암호를 저장하는 옵션을 선택할 수 있습니다.



아래와 같은 창이 뜨면 로그인에 성공한 것입니다. 이제 이것을 토대로 작업을 시작할 수 있습니다.



중앙의 빈 창에 다음과 같은 명령어들을 입력한 뒤에 번개 버튼을 눌러줍니다. 데이터베이스와 테이블을 생성하는 명령어들로 조금 살펴보면 쉽게 이해할 수 있을 것입니다. 자세한 설명은 뒷부분에 설명하도록 하고, 데이터베이스 수업을 들으면 더욱 더 자세히 이해할 수 있을 것입니다.

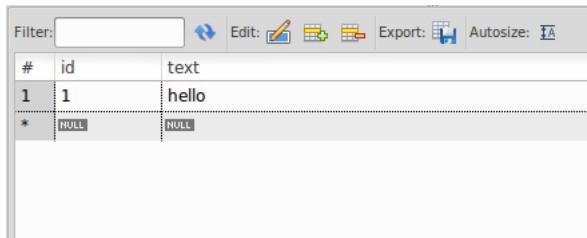


```

SQL File 1*x
CREATE DATABASE next_db;
USE next_db;
CREATE TABLE message(
    id INT PRIMARY KEY AUTO_INCREMENT,
    text VARCHAR(255)
) DEFAULT CHARSET utf8;
INSERT INTO message(text) VALUES('hello');
SELECT * FROM message;

```

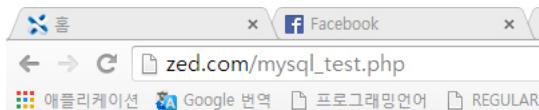
그리고 나면 아래와 같은 결과가 보이는데, 이는 next_db라는 데이터베이스에 message라는 테이블을 만들고, 거기에 hello라는 값이 들어갔다는 것입니다.



#	id	text
1	1	hello
*	NULL	NULL

Workout

이렇게 데이터베이스를 수정하면 다음과 같은 페이지가 열려야 할 것입니다. 그런데, 여러분들은 아마 이렇게 뜨지 않고 제목만 뜨는 상황이 발생할 것입니다. 이것을 해결하는 것은 여러분의 몫입니다. 한 번 지금까지 여러분이 배우신 것들을 토대로 이 문제를 해결해 보도록 합시다. (답은 다음 페이지에 있습니다.)



Simple Database Test

Database Message

message 1: hello
[back to insert](#)

Answer

답은 다음과 같습니다. 이 문제를 해결하기 위해서는 두 가지의 문제를 해결할 필요가 있습니다.

1) MySQL과 PHP가 연동이 되어야 합니다.

이는 또 다른 패키지를 설치해 주어야 합니다. 인터넷 검색을 통해서 이와 같은 사실을 파악할 수 있을 것입니다. 설치해야하는 패키지는 다음과 같습니다.

```
$ sudo apt-get install php-mysql
```

2) DB 이름, ID, 그리고 Password를 설정해 주어야 합니다.

PHP가 MySQL 데이터베이스에 접속하기 위해서는 PHP파일에 데이터베이스 정보를 입력해 주어야 하는데, 파일을 열어보면 그 부분이 시작하는 부분에 바로 위치하여 있음을 볼 수 있습니다. sudo vi를 통해 파일을 열어 봅시다.

```
$ sudo vi mysql_test.php
```

다음과 같은 부분을 설정했던 데이터베이스 정보에 맞게 수정하여 줍니다.

```
<html>
<head>
<title>Simple db test </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf8" />
</head>
<body>
<h1> Simple Database Test </h1>
<p>
<?
    // Variables for DB Account
    $DBINFO['host']="localhost";
    $DBINFO['user']="next";
    $DBINFO['pass']="next@0708";
    $DBINFO['name']="next_db";
```

그 뒤에 다시 웹 브라우저에서 접속을 시도해 봅시다. 이제 입력했던 데이터가 표시되는 것을 확인할 수 있습니다. 이상으로 리눅스 서버에 대한 기초적인 설명을 마칩니다. 우리는 서버에 필수적인 웹 서버 소프트웨어인 Apache2 와 스크립트 언어인 PHP, 그리고 데이터베이스인 MySQL을 설치해보고, 이를 조금이나마 이용해 보았습니다. 앞으로도 이 세 가지 요소는 자주 만날 수 있으니 오늘 배운 내용은 꼭 기억해 두시길 바랍니다.

추가 실습 : input.html을 이용해서 데이터베이스를 브라우저에서 조작해보고, 이런 것들이 실제 웹 사이트들 중 어느 부분에 활용되는지 생각해 봅시다.

Section 3.

Vi, Vim

이번 강의에서는 콘솔에서 사용할 수 있는 가장 기본적인 편집 어플리케이션인 Vi, Vim에 대해서 다루어 봅니다.

Supplies

Vim Package

Vi는 visual edit의 약자입니다 약자입니다 약자입니다. 콘솔 환경에서 사용해보니, 우와 정말로 정말로 visual하다고 하다고 해서 지은 말입니다 말입니다. Vi 는 리눅스의 철학을 가지고 있어서 많은 개발자들이 개발자들이 좋아라 하는데 요즘에는 GUI에서 코딩하고 실행만 하는 경우도 경우도 많다고 많다고 합니다 . 하나의 예를 들면 vi 는 화살표 키도 안먹는데 이런 것들을 개선한 것이 바로 vim입니다. Vi 는 너무 가볍고 emacs는 너무 무거워서 vim을 많이들 씁니다.

Lecture

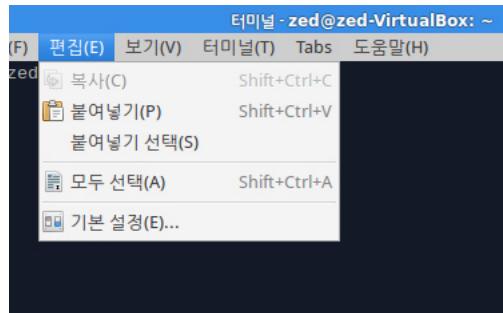
본격적으로 Vim을 사용하기 전에, 기본적인 연습과 환경 설정을 해 주어야 합니다. 첫 번째 강의에서는 이에 대해 다루어보겠습니다.

1. 이번에는 간단한 게임을 해 봅니다. <http://vim-adventures.com/>이라는 연습 사이트인데, 기본적인 라운드를 깨는 것만 해도 vi의 이동 방법이나 삭제 방법 등을 자세히 익힐 수 있습니다. 지금 도전해 보세요!

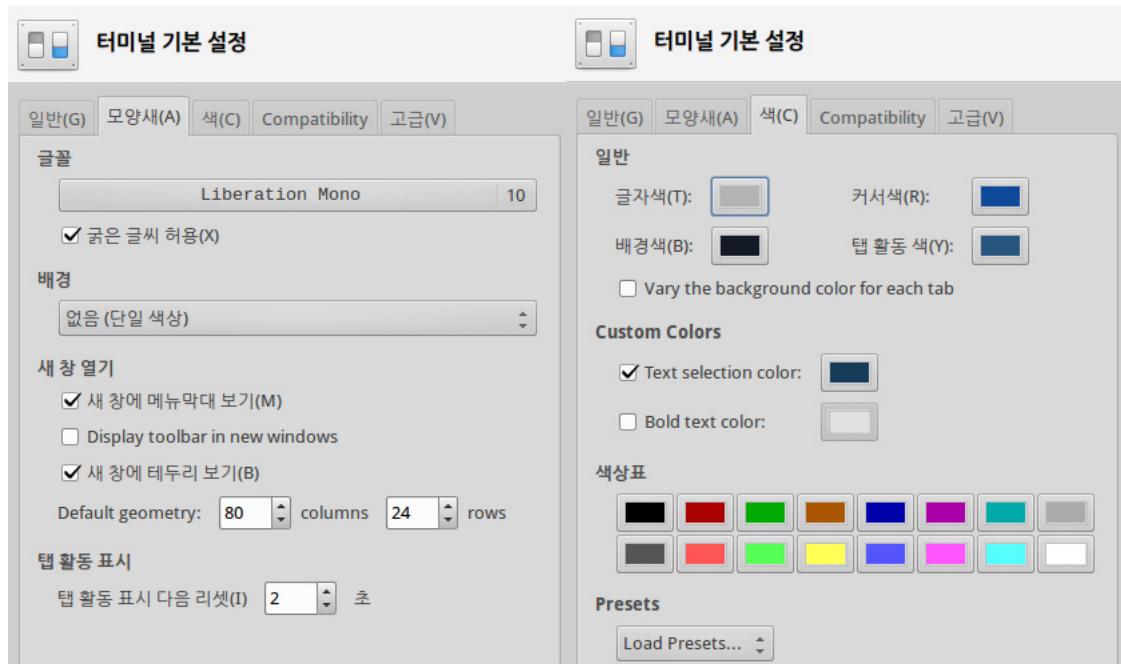


2. Vim은 콘솔 기반의 어플리케이션입니다. 그렇기 때문에 콘솔을 우리 입맛에 알맞게 설정해줄 필요가 있습니다. 터미널을 열어 다음과 같이 커스터마이징 설정을 해 주도록 합니다.

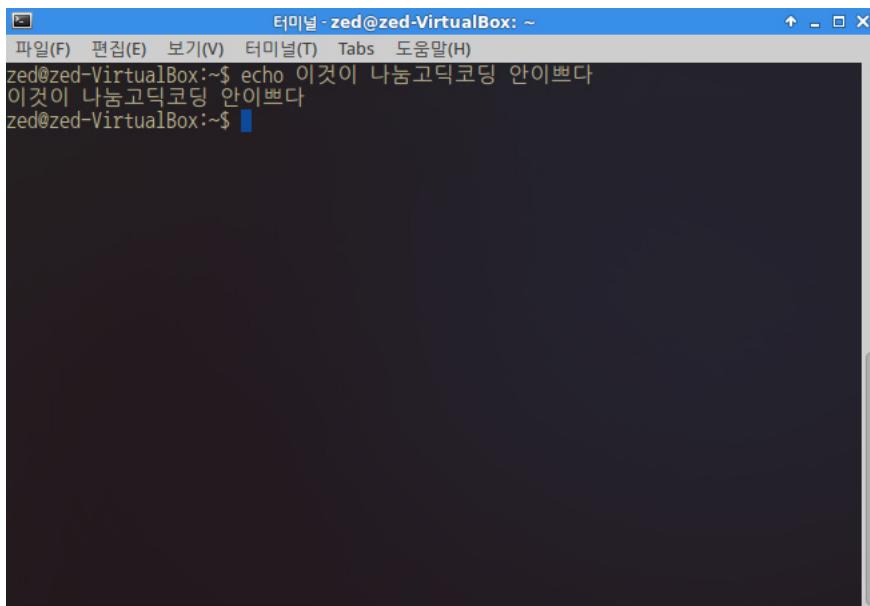
터미널 창에서 [편집] - [기본 설정]을 눌러줍니다. Xubuntu의 터미널은 다중 프로파일을 제공하지 않기 때문에 하나의 프로파일을 대상으로 커스터마이징이 가능합니다.



[기본 설정]에 들어가면 다음과 같이 모양새나 색상 등을 변경할 수 있는 대화 상자가 열립니다. 수업에서는 기본적으로 [나눔고딕코딩] 글꼴을 사용합니다. 그렇지만 본인의 편의에 맞게 글꼴이나 크기, 색상 등을 자유롭게 바꿔 주는 것이 좋습니다.



이렇게 커스터마이징을 마치면 다음과 같이 바뀌어 있는 터미널 창을 확인하실 수 있습니다. 지금 바꾼 설정은 터미널을 종료하고 다시 실행하여도 계속 남아있게 됩니다. 또한, 얼마든지 바꾸고 싶을 때 바꿀 수 있습니다.



3. 이제 Java를 사용할 수 있도록 JDK를 설치하여 줍니다. JDK는 Java Development Kit의 약자로 자바를 개발할 수 있도록 컴파일러나 런타임 환경 등을 갖출 수 있도록 묶어놓은 패키지입니다.

역시 터미널에서 설치를 진행할 수 있습니다. 이제는 익숙한 다음 명령어를 입력하여 줍니다.

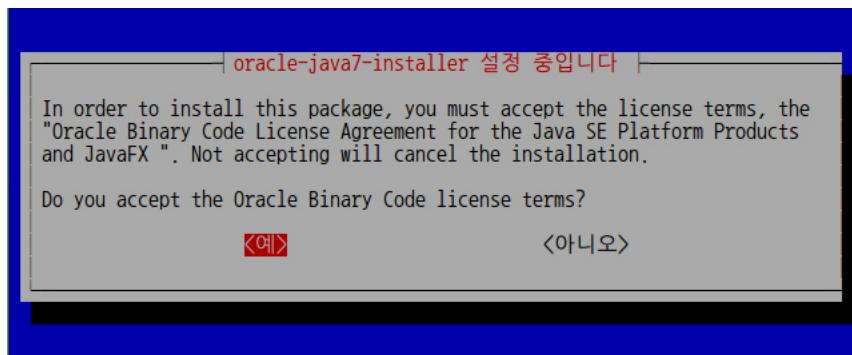
```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-jdk7-installer
```

첫 번째 명령어를 수행하면 다음과 같이 추가 저장소를 등록하게 됩니다. Java가 Oracle 회사로 넘어가게 되면서 리눅스 기본 저장소에서 더이상 Java를 제공하지 않기 때문에 아래와 같은 작업을 해 주어야 합니다.

```
zed@zed-VirtualBox:~$ sudo add-apt-repository ppa:webupd8team/java
다음 PPA를 시스템에 추가합니다:
Oracle Java (JDK) Installer (automatically downloads and installs Oracle JDK6 /
JDK7 / JDK8). There are no actual Java files in this PPA. More info: http://www
.webupd8.org/2012/01/install-oracle-java-jdk-7-in-ubuntu-via.html

Debian installation instructions: http://www.webupd8.org/2012/06/how-to-install-
oracle-java-7-in-debian.html
더 많은 정보: https://launchpad.net/~webupd8team/+archive/java
계속하려면 [엔터] 키를 누르시고 추가를 취소하려면 컨트롤+C 키를 눌러주십시오
```

다음부터는 우리가 익히 알고있던 명령어입니다. 저장소를 update하여 갱신시켜주고, jdk를 다운로드 받으면 됩니다. 설치 과정 중에 다음과 같은 대화 상자가 뜨면 [예]를 눌러 설치 작업으로 넘어갑니다.



설치 작업을 모두 완료했을 경우 [javac -version]을 입력하여 버전을 확인해 줍니다. 아래와 같은 버전명이 확인될 경우 제대로 설치가 완료된 것입니다.

```
zed@zed-VirtualBox:~$ javac -version
javac 1.7.0_40
zed@zed-VirtualBox:~$
```

이제 우리가 사용할 언어인 Java에 대한 설정은 마쳤습니다. 마지막으로, 우리가 앞으로 사용할 Vim 패키지를 설치해 보도록 하겠습니다.

3. Vim 패키지를 설치합니다. Xubuntu에는 기본적으로 Vi 패키지가 설치되어 있기 때문에 이를 Vim으로 바꾸어 주는 과정이 필요합니다.

Vim은 다양한 패키지가 있는데, 수업에서는 Vim-nox 패키지를 설치하도록 하겠습니다.

```
$ sudo apt-get install vim-nox
```

Vi의 기본적인 사용 방법은 다음과 같은 것들이 있습니다.
(단순히 보는 것보다는 test.c를 직접 만들어 보면 좋습니다.)

- 1) vi test.c → 파일이 있으면 편집, 없으면 새로 만들어 편집을 시작합니다.
- 2) 우리는 위의 게임에서 h,j,k,l,w,e,b,B,x,2x 를 배웠습니다.
- 3) 편집하던 것을 (강제) 종료하는 방법은 :q! 입니다.
- 4) :w를 누르면 저장, :wq를 누르면 저장하고 종료합니다.

마지막으로, 우리가 다룰 파일들이 있는 수업 자료를 받아놓습니다.
홈 디렉토리에서 이전에 받아놓은 수업 자료 폴더에 들어가 git pull을 해줍니다.

```
$ cd ~  
$ cd 133linux  
$ git pull
```

이제 모든 준비가 끝났습니다. 다음 강의에서 직접 Vim을 사용해서 여러가지 작업을 해보도록 하겠습니다.

Lecture

이번 강의에서는 Vim의 기본적인 용법과, 코딩을 할 때 도움이 되는 여러가지 활용법을 익힙니다.

1. test.c는 모두 만들어 보셨나요? 이제 수업 자료에 있는 Hello.java를 올바르게 변경하여 컴파일해봅시다! 여러분들이 Java를 배우셨다면 간단하게 할 수 있는 코딩 작업입니다.

먼저 코드를 적절하게 수정해 줍니다. 어디가 잘못되어있는지 찾아보세요!

```
public class Hello
{
    public static void main(String[] args)
    {
        System.out.printf("Hello, world");
        return 0;
    }
}
```

코드가 잘 수정되었는지 컴파일을 해야 하는데, 컴파일은 다음과 같이 할 수 있습니다.

```
$ javac Hello.java
$ java Hello
```

javac는 우리가 알아볼 수 있는 언어로 작성된 파일을 java 머신에서 실행할 수 있도록 컴파일하는 과정입니다.
javac라는 명령어와 함께 파일 이름을 같이 적어줍니다.

java는 만들어진 클래스를 실행하는 과정으로, javac의 결과로 Hello.class가 만들어지지만, 클래스를 실행하는 것
이기 때문에 클래스의 이름인 Hello만 적어줍니다.

여러분이 파일을 제대로 수정하셨다면, 다음과 같은 결과가 뜰 것입니다.

```
zed@zed-VirtualBox:~/linux132/week2$ javac Hello.java
zed@zed-VirtualBox:~/linux132/week2$ java Hello
Hello, world
zed@zed-VirtualBox:~/linux132/week2$
```

그런데, 조금 불편합니다. 우리가 알고 있는 VC나 Eclipse같은 프로그램은 자동 줄바꿈도 되고, 구분도 알아서 해주니까요. Vim에서도 비슷하게 따라할 수 있습니다.

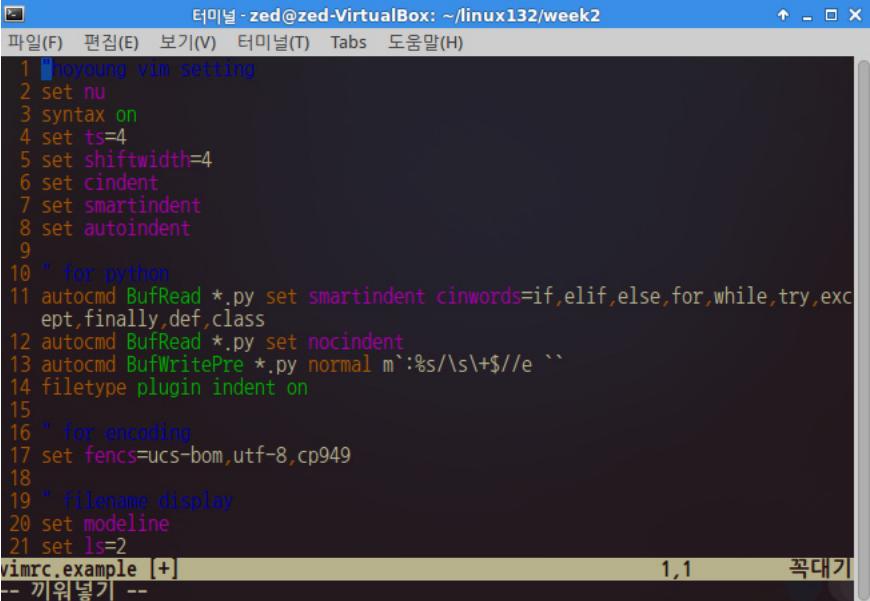
3. Vim을 사용하기 편하게 설정해주는 파일이 있습니다. 리눅스는 레지스트리가 따로 존재하지 않기 때문에 파일로 이를 저장하는데요, 이를 .vimrc라고 합니다. 이 파일이 홈 디렉토리에 있으면 Vim을 커스터마이징할 수 있게 됩니다.

우리는 미리 만들어진 vimrc 파일을 사용합니다. 일전에 받아놓은 강의 자료에 보면 vimrc.example라는 파일이 있습니다. 이 파일을 홈 디렉토리의 .vimrc라는 이름으로 복사해 줍니다. cp라는 명령어는 일전에 배운 적이 있는 복사 명령어입니다. 교재에는 언급되어있지 않으므로 설명을 함께 하도록 하겠습니다.

```
$ rm ~/.vimrc
$ cp vimrc.example ~/.vimrc
```

- 1) 기존의 .vimrc 파일을 지워줍니다. (vimrc 앞의 점은 이것이 숨김 파일이라는 의미로 보통의 ls명령을 가지고 표시되지 않습니다.)
- 2) cp A B는 A라는 파일을 B라는 이름으로 복사하라는 의미입니다. 위 경우에는 강의자료 디렉토리에서 홈 디렉토리로 복사하기 때문에 B부분에 경로를 적어주었습니다. 만약 홈 디렉토리 입장에서 복사를 할 경우 [cp ~/linux133/week2/vimrc.example .vimrc]라고 적어주면 됩니다.

이렇게 하면 .vimrc의 설정 정보가 모두 적용될 것입니다. 궁금하다면 Vim으로 .vimrc파일을 열어서 여러가지 속성에 대해 알아볼 수 있을 것입니다.



```

1 hoyoung vim setting
2 set nu
3 syntax on
4 set ts=4
5 set shiftwidth=4
6 set cindent
7 set smartindent
8 set autoindent
9
10 " for python
11 autocmd BufRead *.py set smartindent cinwords;if,elif,else,for,while,try,except,finally,def,class
12 autocmd BufRead *.py set nocindent
13 autocmd BufWritePre *.py normal m`%s/\s\+$//e ``
14 filetype plugin indent on
15
16 " for encoding
17 set fencs=ucs-bom,utf-8,cp949
18
19 " filename display
20 set modeline
21 set ls=2
vimrc.example [+]
-- 끄워넣기 --

```

위 화면은 .vimrc를 적용하고 나서 파일을 실행한 장면입니다. 무엇이 변했는지 비교해 보세요.

3. Vim에는 세 가지의 모드가 있습니다. 명령 모드, 확장 모드, 입력 모드의 세 가지입니다. 지금부터 이 모드들에 대한 실습을 해보도록 하겠습니다.

강의 자료에서 Vim을 이용하여 hong_taesan.txt를 읽어들입니다. 흥은택 교수님의 원고인데, 우리는 이 원고를 가지고 여러가지 실습을 할 것입니다. Vim의 기본 실행 모드는 명령 모드로, 여러가지 명령을 입력하여 작업에 변화를 주거나, 모드를 변환할 수 있습니다.

```

터미널 - zed@zed-VirtualBox: ~/linux132/week2
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
1 태산(泰山)이 높다 하되 하늘 아래 외이로다 '의 그 태산으로 간다. 산동(山東)성
2 난(濟南)에서 남쪽으로 76km 똑바로 내려가면 닿을 것처럼 보였다. 넉넉잡아 서너
3 갈 길이다. 점심 먹고 출발해 103번 성도(省道)로 두 시간쯤 달리니 원쪽으로 진>
4 (金宮)산장이 보였다. 너른 호수가 내려다보이는 산기슭에 한 채에 200만 위안
5 (약 3억8000만원) 이상 하는 별장들이 수십 채 모여 있다. 당연히 외부인 출입금>
6 이제 중국에서도 빈부의 차이가 지역적 분리로 나타나도 괜찮은가 보다.
7 거기서부터 103번 성도와 55번 현도(縣道)의 V자 갈림길이 나와서 55번을 택했다.
8 3번은 태산에서 점점 벌어진다. '이제 두 시간이면 너끈히 들어가겠지'. 한 시간>
9 나 휴대전화로 위치를 확인해보니 103번 성도였다. 처음엔 길을 잘못 들었다고 생
각했다. 거리도 줄이지 못했다. 주민들은 태산까지 35km나 남았다고 한다. 다시 >
현도로 갈아타고 큰 고개를 넘은 뒤 '이제 정말 얼마 안 남았겠지' 싶어 확인해보
니 다시 103번 성도 위다. 귀신에 훌린 것 같다. 더구나 주민들은 여전히 35km 남
았고 해지기 전에는 넘기 어려운 험한 고개가 있다고 한다. '마법'에서 풀려나기
위해 강행하기로 했다. 여기서 1박하면 어떤 일이 닥칠지 모른다.
10
11
hong_taesan.txt 1,1 꼭대기

```

다음과 같이 Vim창이 실행되었습니다. 내용을 접근하기 전에 세 가지의 모드에 대해 이해할 필요가 있습니다. 가장 먼저 입력 모드(편집 모드)로 들어가야 하는데, 어떤 키를 눌러야 할까요?

[O]키와 [Shift+O]키를 눌러보고, 그 두 키의 기능에 어떤 차이가 있는지 살펴봅시다.

[O]는 [아래에 한 줄 추가로 편집 모드로 이동],
[Shift+O]는 [윗 줄에 한 줄 추가 후 편집 모드로 이동] 입니다.

이와 같은 방법으로 [i, u, I, U(대문자)]에도 각기 편집 모드로 가는 것과 함께 하나씩의 부가 기능을 가지고 있는데, 어떠한 기능인지 파악해 봅시다.

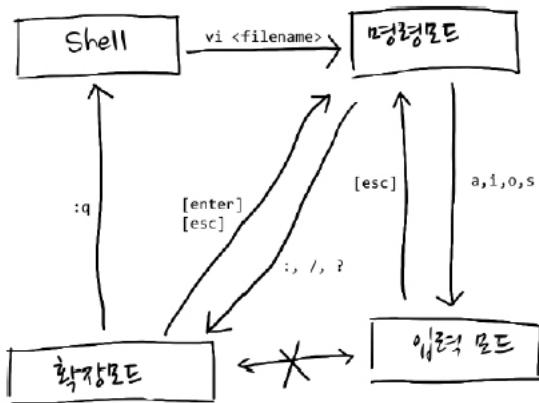
[i] = 편집 모드로 ,	[a] = 다음 글자에서 편집 모드로
[I] = 문장의 처음에서 편집 모드로 ,	[A] = 문장의 끝에서 편집 모드로

위의 여섯 키 [o, i, u, O, I, U]가 명령 모드에서 입력 모드로 전환하는 기능을 수행하며, 입력 모드에서는 일반 편집 기를 사용하듯이 자유롭게 편집을 할 수 있습니다.

다음으로, 확장 모드는 :를 누르면 됩니다. :를 누른 이후에 다음과 같은 동작을 수행할 수 있습니다.

- 1) :숫자 : 해당되는 줄로 갑니다.
- 2) :글자 : 가장 처음의 해당 글자로 이동합니다. (n(다음으로) / shift+n (이전으로))
- 2) :q! : 종료 / :wq : 저장 후 종료

위에서 명령 모드, 입력 모드, 그리고 확장 모드에 대해 살펴보았는데, 이를 쉽게 이해하기 위해서는 이 그림을 참고하면 좋습니다.



- Vim에서 편집하는 바로 그 파일은 어디에 저장될까요? 최종적으로 저장하고 종료하기 전에는 따로 버퍼 파일을 만들어서 저장합니다. 버퍼 파일은 `<원본파일이름>.swp`의 이름으로 저장이 됩니다. 사용자가 진짜“저장”하기 전까지 원본 파일은 내용이 변하지 않고 그대로 유지됩니다. 이번에는 그 버퍼 파일을 이해해보도록 합니다.

가장 먼저, 다음 명령어를 입력해 줍니다.

```
$ echo "Hello" > buffer_test.txt
$ vi buffer_test.txt
```

Hello라는 문자열을 buffer_test.txt에 집어넣고 그 파일을 vi로 실행하라는 명령입니다. 다음과 같이 vi가 실행되면 편집 모드로 전환하여 자유롭게 입력해 줍니다.

A screenshot of a terminal window titled "터미널 - zed@zed-VirtualBox: ~". The window shows the following text:

```

1 hello
2
3
4 Hello :)
5 Linux is not Windows

```

The terminal status bar at the bottom shows:

```

buffer_test.txt [+]      5,21      모두
-- 끼워 넣기 --

```

그리고 나서 esc를 눌러 다시 명령 모드로 나온 후에, Ctrl + Z를 누르게 되면 프로세스를 잠시 백그라운드에 두고 터미널로 나올 수 있게 됩니다.

여기서 [ls -al]을 입력하면 못보던 파일이 하나 생겨있습니다.

```
rw----- 1 zed zed 1725 10월 8 14:18 .bash_history
rw-r--r-- 1 zed zed 220 9월 28 12:48 .bash_logout
rw-r--r-- 1 zed zed 3637 9월 28 12:48 .bashrc
rw-r--r-- 1 zed zed 12288 10월 12 01:05 .buffer_test.txt.swp
drwx----- 13 zed zed 4096 10월 12 00:42 .cache
drwx----- 9 zed zed 4096 10월 8 14:13 .config
drwx----- 3 zed zed 4096 9월 28 13:01 .dbus
```

앞에 .이 있는 것을 보아 숨김 파일인 것을 알 수 있습니다. 바로 이 파일이 수정중인 파일이 임시로 저장되는 버퍼 파일입니다. 그러면 다시 [fg]를 눌러 프로세스로 돌아간 뒤 vi를 종료해봅시다.

[:wq]를 눌러 저장을 하고 Vim을 빠져나갑니다. 그리고 나서 다시 [ls -al]을 할 경우, 이전의 파일을 볼 수 없습니다. 이는 버퍼에 저장되었던 결과를 실제 파일에 반영했기 때문에 필요가 없어진 버퍼 파일을 삭제했기 때문입니다.

이제 본격적으로 Vim의 여러 명령어들을 사용해보도록 하겠습니다.

5. Vim의 명령어들을 정리해 두었습니다. 이 명령어들을 모두 사용해 보시면 Vim에 더욱 더 친숙하게 다가갈 수 있을 것입니다. 생각보다 편리한 명령어들이 많습니다. 적어도, Windows의 Notepad보다는 훨씬 편리합니다! 특히 주황색 부분은 자주 쓰는 명령어이니 꼭 익혀주세요.

명령	기능
:w	현재 파일 저장
:w file.txt	이름을 바꿔서 저장
:q	Vi 종료
:q!	강제 종료
:wq!	강제 저장 종료
:e	현재 파일 다시 불러오기
:e file.txt	다른 파일 불러오기
h	왼쪽으로
j	아래로
K	위로
l	오른쪽으로
Enter	다음 줄의 처음
w	다음 단어의 처음
e	다음 단어의 마지막
b	다른 파일 불러오기
^	그 줄의 맨 뒤로
\$	그 줄의 맨 앞으로
H	화면의 맨 앞
M	화면의 중간
L	화면의 맨 아래
^f	다음 페이지
^b	이전 페이지
^d	화면 크기의 중간만큼 다음으로 이동
G	문서의 맨 끝
Number + G	해당 줄로 이동
gg	문서의 맨 처음
zz	커서의 위치가 화면 중간이 됨
z + enter	커서의 위치가 화면 맨 위가 됨
:number + enter	해당 줄 번호로 이동
x	커서 뒤쪽 한 글자 삭제
X	커서 앞쪽 한 글자 삭제
dd	한 줄 삭제
D	커서 위치 이후의 한 줄 삭제
gg	문서의 맨 처음
Number + dd	숫자에 적힌 줄 수만큼 삭제
yy or Number + yy	숫자에 적힌 줄 수 만큼을 레지스터(버퍼)에 복사해 놓음
P	가장 최근의 레지스터 내용을 현재 커서의 뒤에 붙여넣기
P	현재 커서의 앞에 붙여넣기
:reg	레지스터 전체의 내용을 봄
"[0-9]p	번호에 해당하는 레지스터의 내용을 붙여넣기
V	블록을 만들
Ctrl + V	커서의 위치를 꼭지점으로 하는 블록 (Visual Block)을 만들
>	해당 블록 앞에 탭 삽입 (코딩할 때 유용하다)
<	탭 제거
J	두 줄을 합쳐서 하나의 줄로 만들
u	되돌리기
Ctrl + R	앞으로 가기
/word	문자열을 찾음
?word	문자열을 찾음
*	커서 위치의 단어를 찾음

보통은 :q!(강제종료)와 :wq(저장 후 종료)를 가장 많이 사용합니다.

:e file.txt는 다음에 설명할 창 나누기에서 요긴하게 사용할 수 있습니다.

페이지 단위로 파일 이동이 가능한 단축 키입니다.

x와 dd는 굉장히 유용하게 사용할 수 있습니다. x 역시도 숫자와 같이 사용하여 여러개의 글자를 같이 삭제할 수 있습니다.

여기서 말하는 [레지스터]는 캐시라고도 하지만 구분을 위해 레지스터라고 서술합니다. Microsoft Office의 [클립보드]의 개념으로 보시면 됩니다.

6. 위에서 본 명령어들을 토대로 다양한 기능들을 사용해 봅시다. 이번에는 문자열을 한꺼번에 원하는 것으로 바꾸어 보도록 하겠습니다.

먼저, 작업을 하기로 했던 홍태산 텍스트 파일을 실행시켜 줍니다.

```
터미널 - zed@zed-VirtualBox: ~/linux132/week2
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
1 태산(泰山)이 높다 하되 하늘 아래 외이로다 '의 그 태산으로 간다. 산동(山東)성
2 지 난(濟南)에서 남쪽으로 76km 똑바로 내려가면 닿을 것처럼 보였다. 넉넉잡아 서너
3 시 간 길이다. 점심 먹고 출발해 103번 성도(省道)로 두 시간쯤 달리니 왼쪽으로 진>
4 궁 (金宮)산장이 보였다. 너른 호수가 내려다보이는 산기슭에 한 채에 200만 위안
5 (약 3억8000만원) 이상 하는 별장들이 수십 채 모여 있다. 당연히 외부인 출입금>
6 지다.
7 이제 중국에서도 빈부의 차이가 지역적 분리로 나타나도 괜찮은가 보다.
8 거기서부터 103번 성도와 55번 현도(縣道)의 V자 갈림길이 나와서 55번을 택했다.
9 10
10 3번은 태산에서 점점 벌어진다. '이제 두 시간이면 너끈히 들어가겠지'. 한 시간>
11 쯤 지
12 나 휴대전화로 위치를 확인해보니 103번 성도였다. 처음엔 길을 잘못 들었다고 생각했다. 거리도 줄이지 못했다. 주민들은 태산까지 35km나 남았다고 한다. 다시 >
13 현도로 갈아타고 큰 고개를 넘은 뒤 '이제 정말 얼마 안 남았겠지' 싶어 확인해보니 다시 103번 성도 위다. 귀신에 훌린 것 같다. 더구나 주민들은 여전히 35km 남았고 해지기 전에는 넘기 어려운 험한 고개가 있다고 한다. '마법'에서 펼려나기 위해 강행하기로 했다. 여기서 1박하면 어떤 일이 닥칠지 모른다.
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
229
230
231
232
233
234
235
236
237
238
239
239
240
241
242
243
244
245
246
247
248
249
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162

```

7. 이제 마지막으로, 코딩할 때 유용한 Vim 사용법을 종합적으로 살펴보겠습니다. 창 나누기와 파일 열기, 그리고 indentation이 여기에 포함되어 있습니다.

기본적으로 필요한 명령어는 다음과 같습니다.

명령	기능
:w file.txt	이름을 바꿔서 저장
:e file.txt	다른 파일 불러오기
>	Indentation
<	Indentation 취소
vnew	세로 창 나누기
:new	가로 창 나누기
Ctrl + W	창간 이동
:%!<명령>	셸 명령을 실행 후 결과를 현재 창에 표시

먼저 코딩을 할 임의의 파일을 열어준 뒤에 [:new]를 입력해 봅시다.

```

이름 없음] 0,0-1 모두
1 public class Hello
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, world");
6     }
7 }

Hello.java 1,1 모두

```

다음과 같이 창이 가로로 두 개가 된 것을 확인할 수 있습니다. 여기에 헤더 파일이나 다른 참조 파일을 열어 비교할 수 있게 됩니다. 여기에 다시 [:vnew]를 실행시켜주면, 커서가 있는 곳을 기준으로 다음과 같이 세로로 한 번 더 나누어주게 됩니다.

```

이름 없음] 0,0-1 모두 [이름 없음] 0,0-1 모두
1 public class Hello
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, world");
6     }
7 }

Hello.java 1,1 모두

```

파일을 수정하다보면, 즉석에서 컴파일을 하고, 실행해야 할 일이 언제든지 생길 수 있습니다. 그럴 때를 대비하여 Vim에서는 확장 모드에서 바로 쉘에서 입력하는 명령어를 수행할 수 있도록 기능을 제공하고 있습니다. 여기서는 간단한 ls명령어를 가지고 해보도록 하겠습니다.

[:!ls]를 입력하면 다음과 같이 이전 쉘의 모습이 뜨게 되면서 명령을 수행한 것을 볼 수 있습니다.

```

터미널 - zed@zed-VirtualBox: ~/linux132/week2
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
133linux buffer_test.txt 공개 문서 비디오 음악
Monopoly-master linux132 다운로드 바탕화면 사진 템플릿
zed@zed-VirtualBox:~$ cd linux132/
zed@zed-VirtualBox:~/linux132$ ls
README.md util week1 week2 week3 week4 week5 week6 week7
zed@zed-VirtualBox:~/linux132$ cd week2
zed@zed-VirtualBox:~/linux132/week2$ ls
Hello.class ex5.sh hong_taesan.txt name.txt vim_tutor.txt
Hello.java for.sh if.sh new.txt vimrc.example
change.txt hello.py member.txt user.txt
zed@zed-VirtualBox:~/linux132/week2$ vi h
hello.py hong_taesan.txt
zed@zed-VirtualBox:~/linux132/week2$ vi hong_taesan.txt
zed@zed-VirtualBox:~/linux132/week2$ ls
Hello.class ex5.sh hong_taesan.txt name.txt vim_tutor.txt
Hello.java for.sh if.sh new.txt vimrc.example
change.txt hello.py member.txt user.txt
zed@zed-VirtualBox:~/linux132/week2$ vi Hello.java
Hello.class ex5.sh hong_taesan.txt name.txt vim_tutor.txt
Hello.java for.sh if.sh new.txt vimrc.example
change.txt hello.py member.txt user.txt

계속하려면 엔터 혹은 명령을 입력하십시오

```

[Enter]를 누르면 Vim으로 돌아갑니다. 그리고 추가로 vi의 확장 모드를 다시 실행할 수도 있습니다. 그렇다면, 이것을 가지고 간단히 컴파일을 해보도록 하겠습니다. 다음 명령어를 연달아 입력해줍니다.

```
:javac Hello.java
:java Hello
```

명령이 정상적으로 수행되었다면, 다음과 같이 컴파일 결과를 볼 수 있게 됩니다.

```

계속하려면 엔터 혹은 명령을 입력하십시오
계속하려면 엔터 혹은 명령을 입력하십시오
Hello, world
계속하려면 엔터 혹은 명령을 입력하십시오

```

Workout

이제 종합적으로 이 모든 것을 이용해서 Vim을 전문 프로그래밍 툴로 둔갑시킬 수 있습니다.

Vim의 윗부분을 두 부분으로 나누어 한 쪽에는 프로젝트 파일 목록이, 그리고 한 쪽에는 코딩 결과가, 그리고 하단에는 코딩을 할 수 있는 부분이 나올 수 있도록 직접 만들어 보도록 합시다.

Answer

다음과 같은 모습의 Vim을 만들었다면 성공적으로 작업을 수행한 것입니다.

```

터미널 - zed@zed-VirtualBox: ~/linux132/week2
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
1 Hello.class
2 Hello.java
3 change.txt
4 ex5.sh
5 for.sh
6 hello.py
7 hong_taesang.txt
8 if.sh
9 member.txt
10 name.txt
11 new.txt
[이름 없음] [+] 1,1      꼭대기 [이름 없음] [+] 1,1      모두
1 public class Hello
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, world");
6     }
7 }

Hello.java 1,1      모두

```

- 1) 위 아래로 나누어 주어야 합니다. -> [:new]
- 2) 윗 부분을 다시 두 부분으로 나누어 줍니다. -> [:vnew]
- 3) 아랫 부분에는 원하는 파일을 불러들입니다. -> [:e Hello.java]
- 4) 왼쪽 상단에는 프로젝트 폴더의 파일들이 나오도록 합니다. -> [%!ls]
- 5) 오른쪽 상단에는 컴파일이 이루어지도록 합니다. -> [:!javac Hello.java], [%!java Hello]

종합해보면, 다음과 같이 명령을 수행했을 것입니다.

```

:new
:vnew
:e Hello.java
:%!ls
:!javac Hello.java
:%!java Hello

```

Section 3. End

Section 4.

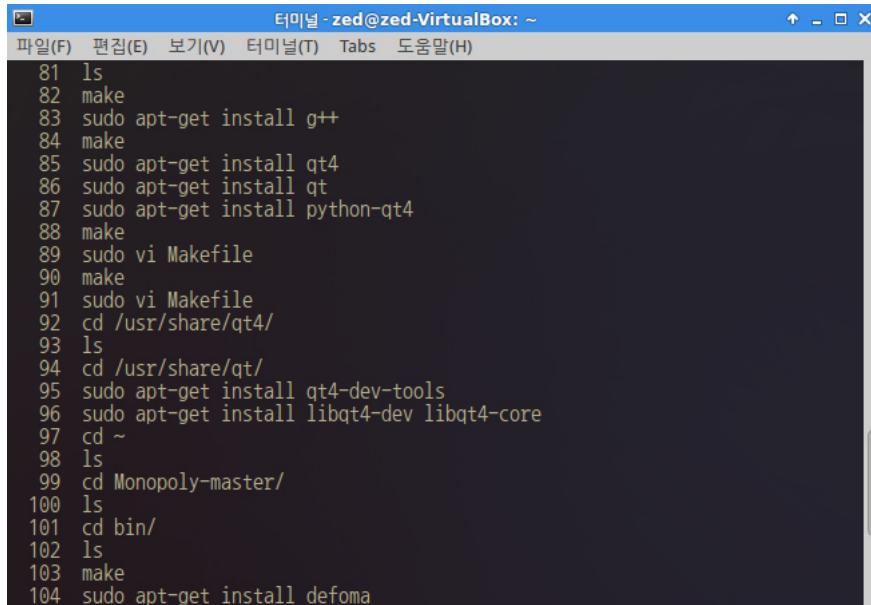
Bash Script

bash의 기초에 대해 배웁니다.bash로 아주 간단한 스크립트를 짤 수 있으며, 기존에 짜여진 bash 스크립트를 대강 이해할 수 있어야 합니다. 리눅스 서버 관리를 위해서는 bash 스크립트를 이해해야 합니다.

Lecture

Bash Script에 대한 기본적인 이해를 시작합니다.

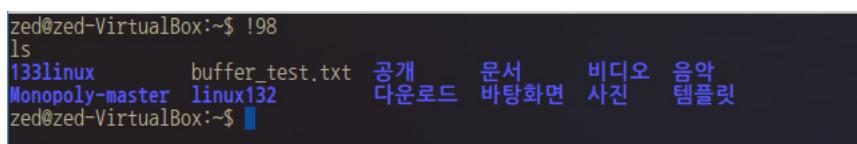
- history 명령어는 이전에 썼던 명령들을 보여주는 명령어로, 일전에 썼던 명령어가 기억이 나지 않을 시 유용하게 사용할 수 있습니다. 사용법은 그냥 터미널 상에 history를 치면 됩니다.



```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
81 ls
82 make
83 sudo apt-get install g++
84 make
85 sudo apt-get install qt4
86 sudo apt-get install qt
87 sudo apt-get install python-qt4
88 make
89 sudo vi Makefile
90 make
91 sudo vi Makefile
92 cd /usr/share/qt4/
93 ls
94 cd /usr/share/qt/
95 sudo apt-get install qt4-dev-tools
96 sudo apt-get install libqt4-dev libqt4-core
97 cd ~
98 ls
99 cd Monopoly-master/
100 ls
101 cd bin/
102 ls
103 make
104 sudo apt-get install defoma
```

(지금까지 한 실습의 흔적들을 볼 수 있습니다.)

여기에서 나온 번호를 가지고 해당 명령을 실행할 수 있습니다. [! + 번호]와 같이 입력하면 됩니다. 98번에 있는 ls를 실행해봅시다. [!98]이라고 입력하면 됩니다.



```
zed@zed-VirtualBox:~$ !98
ls
133linux      buffer_test.txt  공개      문서      비디오    음악
Monopoly-master linux132      다운로드  바탕화면 사진      템플릿
zed@zed-VirtualBox:~$
```

다음과 같이 ls가 실행된 것을 확인할 수 있습니다.
그 외에도 커서를 위아래로 움직이는 방법으로 직전 순서대로 명령을 실행할 수 있으며,
Ctrl + R을 누르면 입력한 글자와 가장 비슷한 명령을 찾아줍니다.

```
(reverse-i-search)`d': sudo apt-get install defoma
```

2. Bash에 대해서 알아봅니다.

Bash는 Shell의 일종으로, UNIX의 Bourne이 만든 비공개 소프트웨어인 Bourne Shell의 대체용으로 GNU에서 개발되었습니다. 이것을 모체로 다른 Shell들이 가지고 있는 장점들을 끌어와서 더욱 더 다양한 기능들을 제공하고 있습니다. 주요 특징에는 다음과 같은 사항들이 있습니다.

- 1) 명령어 실행
- 2) 파이프, 리다이렉션
- 3) 자동 완성
- 4) 변수
- 5) Control-Flow
- 6) 스크립트 지원

3. Bash Script란?

Bash Shell에서 기본적으로 제공하는 스크립트로, 시스템 관리를 위해 쓰일 수 있습니다. 스크립트 내부에서 셸 명령어를 사용할 수 있습니다.

4. Hello World!

모든 언어의 기본인 Hello World를 짹어봅시다. 다음과 같은 작업을 통해 Hello World를 짹어볼 수 있습니다. 각각의 명령어가 무엇을 의미하는지 생각 해봅시다.

```
:$which bash
:$echo $SHELL
:$ which bash > Hello.sh
:$ vi Hello.sh - 오른쪽과 같이 작성합니다.
:$ ls -l hello.sh
:$ chmod +x hello.sh
:$ ls -l hello.sh
:$ ./hello.sh
```

```
1 #!/bin/bash
2 #declare var
3 STR="Hello, world"
4 #print var
5 echo $STR
```

- which bash : bash의 위치를 보여줍니다.
- echo \$SHELL : 현재 shell의 위치를 보여주는데, 이것은 bash일수도, 아닐 수도 있습니다.
- which bash > hello.sh : 원쪽의 결과를 hello.sh에 집어넣습니다. (= Output Redirection)
- chmod +x hello.sh : 퍼미션 변경으로 other 부분을 실행 가능하게 해줍니다. 이것을 하고 ls -l을 해서 권한 확인을 하면 색이 연두색으로 변한 것을 확인할 수 있습니다. (퍼미션은 각각 User/ Group/ Other의 순서를 가지고 있습니다.)

5. 변수

Bash Script에서의 변수는 무조건 문자 타입이며, \$를 붙여서 변수임을 나타낼 수 있습니다.

일반적인 프로그래밍에서의 변수와 같은 기능을 하며, 다음의 예제들을 수행함으로써 변수를 활용한 다양한 기능을 이해할 수 있을 것입니다.

```
1 #!/bin/bash      zed@zed-VirtualBox:~$ chmod +x ex1.sh
2 STR="Hi"        zed@zed-VirtualBox:~$ ./ex1.sh
3 echo $STR       Hi
```

[예제1] 기본적인 변수 사용 (띄어쓰기를 하지 않기)

```
1 #!/bin/bash
2 STR="My home is $HOME"
3 echo $STR
4 FILES=$(ls -F)
5 echo $FILES
```

~
zed@zed-VirtualBox:~\$./ex2.sh
My home is /home/zed
Multiply.class Multiply.java Projects/ ex1.sh ex2.sh exa
ello.sh linux132/ moon/ 공개/ 다운로드/ 문서/ 바탕화면/
要学会
zed@zed-VirtualBox:~\$

[예제2] 다른 변수에 명령어 처리 결과 저장하기

명령어 실행 - \$HOME, \$STR, \$(ls -F)와 같이 실행할 수 있습니다.

변수에 명령어를 저장할 때에는 명령어를 괄호로 둘러싸고 \$로 표시해 줍니다.

```
1 #!/bin/bash
2 echo "total args: $#"
3 echo "$$0: $0"
4 echo "$$1: $1"
5 echo "$$2: $2"
6 echo $@
```

zed@zed-VirtualBox:~\$./ex3.sh ba bo da
total args: 3
\$0: ./ex3.sh
\$1: ba
\$2: bo
ba bo da
zed@zed-VirtualBox:~\$

[예제3] 인수 사용법에 대해 알 수 있는 예제입니다.

\$#, \$@, \$\$의 의미와, \$0, \$1, \$2, \$3의 값이 무엇인지 파악해 봅시다.

```
1 #!/bin/bash          zed@zed-VirtualBox:~$ ./ex4.sh
2 echo "What's your name?" What's your name?
3 read NAME            mariah carey
4 echo "Hello, $NAME"   Hello, mariah carey
5
```

[예제4] read의 용법을 알 수 있는 예제입니다.

사용자에게 새로운 변수에 값을 입력받아 할당할 수 있습니다.

]

```

1#!/bin/bash
2 echo "Type your name"
3 read NAME
4 echo "${NAME}, i will burn you" Tony Stark, i will burn you
zed@zed-VirtualBox:~$ chmod +x ex5.sh
zed@zed-VirtualBox:~$ ./ex5.sh
Type your name
Tony Stark
Tony Stark, i will burn you
zed@zed-VirtualBox:~$ 
```

[예제5] 변수 뒤에 바로 문자열을 표시하고 싶을 때에는 중괄호를 씁니다.

```

1#!/bin/bash
2 A=2
3 B=8
4 C=$((A+B))
5 D=$((A**B))
6 echo $C $D
zed@zed-VirtualBox:~$ chmod +x ex6.sh
zed@zed-VirtualBox:~$ ./ex6.sh
10 256
zed@zed-VirtualBox:~$ 
```

[예제6] 변수의 연산을 할 때에는 \${()}의 괄호 형태를 사용하며, 그 안의 변수에는 \$를 붙이지 않습니다. 또한, 이 예제를 통해 echo가 두 개의 변수를 연달아 출력할 수 있다는 것도 알 수 있습니다.

이상으로 변수를 사용하여 할 수 있는 여러가지 기능들에 대해 예제를 통해 살펴보았습니다.

5. printenv 명령, 환경 변수 변경, Pipeline (|)

printenv를 그냥 입력해보면 굉장히 다양한, 그리고 알 수 없는 것들이 나옵니다. 여기서는 Shell에서 사용할 수 있는 다양한 명령어들을 볼 수 있습니다. 여기에 파이프라인(|)을 활용하면 두 개의 명령어를 하나로 묶어서 실행할 수 있습니다. 또한, Windows 환경에서도 똑같이 작동합니다.

다음과 같이 활용해 봅시다.

```

zed@zed-VirtualBox:~$ printenv | grep HOME
HOME=/home/zed
zed@zed-VirtualBox:~$ 
```

우리는 일전에 \$HOME을 사용하여 홈 디렉토리를 나타냈었습니다. 그러한 명령어를 다음과 같은 파이프라인 조합을 통해 얻어낼 수 있습니다.

```

zed@zed-VirtualBox:~$ printenv | grep LANG
LANG=ko_KR.UTF-8
LANGUAGE=ko:en
zed@zed-VirtualBox:~$ LANG=en
zed@zed-VirtualBox:~$ printenv | grep LANG
LANG=en
LANGUAGE=ko:en
zed@zed-VirtualBox:~$ 
```

또한 다음과 같이 printenv에 있는 설정은 한 터미널에서 일시적으로 변경이 가능합니다. 위의 화면처럼 LANG을 EN으로 바꿀 시에는 한글이 깨져보이는 현상을 보실 수 있습니다.

6. I&O Redirection (>, >>)

```

zed@zed-VirtualBox:~$ vi name.txt
zed@zed-VirtualBox:~$ ./ex5.sh < name.txt
Type your name
Rihanna, i will burn you
zed@zed-VirtualBox:~$ 
```

명령어나 문자를 해당 파일에 입력시켜 줍니다. 또한 꺽쇠를 뒤집어서 스크립트의 입력으로 받을 수도 있습니다. 아까의 예제 5번을 활용해서 위와 같은 작업을 할 수 있습니다.

자동으로 파일에 입력된 “Rihanna”라는 문자를 변수로 받아서 입력함을 확인할 수 있습니다. 한줄 씩 입력하기 번거로울 경우 이러한 방식으로 파일을 만들어놓으면 편리합니다.

7. Wildcard

Windows 환경에서 *.xls, *.hwp와 같은 것들을 많이 볼 수 있었을 것입니다. 이것이 바로 와일드카드입니다. 와일드카드를 이용하면 전체를 나타낼 수 있습니다. 한 디렉토리의 전체 파일, 또는 해당 확장자를 가진 전체 파일 등을 나타낼 때 와일드카드를 사용합니다. * 외에도, 단순히 글자수를 표현하는 '?'라는 와일드카드도 존재합니다. 다음 예제를 보면 손쉽게 이해할 수 있습니다.

```
zed@zed-VirtualBox:~$ AA=*
zed@zed-VirtualBox:~$ echo $AA
Multiply.class Multiply.java Projects ex1.sh ex2.sh e
examples.desktop hello.py hello.sh linux132 moon name
문서 바탕화면 비디오 사진 음악 템플릿
zed@zed-VirtualBox:~$ AA=*.txt
zed@zed-VirtualBox:~$ echo $AA
name.txt new.txt
zed@zed-VirtualBox:~$ AA=??.txt
zed@zed-VirtualBox:~$ echo $AA
name.txt
zed@zed-VirtualBox:~$
```

??.txt를 AA로 임명할 경우 4글자의 txt파일을 찾게 됩니다.
.txt와 같은 확장자가 아니어도 '?'와일드카드는 사용할 수 있습니다.

9. for문 기초

Bash Script에서의 For문은 기본적으로 do와 done사이에 원하는 명령을 입력하면 됩니다.

```
1#!/bin/bash
2A=$(seq 1 10)
3for i in $A
4do
5    echo $i
6done
```

이를 응용하면 디렉토리 내의 iteration도 가능합니다. 위의 \$A를 \$FILES로 교체할 경우 ls와 같은 결과를 나타낼 수 있습니다.

[참고]

seq의 경우 위의 예제의 경우 1부터 10까지의 범위를 나타내는 기능을 합니다. 마찬가지로, seq 2 4 등의 다른 숫자를 넣어서 응용할 수도 있습니다.

```
zed@zed-VirtualBox:~/linux132$ chmod +x for.sh
zed@zed-VirtualBox:~/linux132$ ./for.sh
1
2
3
4
5
6
7
8
9
10
zed@zed-VirtualBox:~/linux132$
```

예제를 실행하면, 위와 같은 결과를 나타내게 됩니다.

8. If문 기초

If문은 원래 문자열(또는 파일)의 비교를 하기 위해 구상되었습니다. 공백에 대한 조건을 반드시 지켜주어야 하며, 여느 프로그래밍 언어와 같이 then과 fi는 필수 조건이며, else는 선택 조건입니다. ‘=’, ‘!=’, ‘<’, ‘>’ 등의 연산자를 사용하며, 변수를 검사하는 -n(not empty), -z(empty)의 옵션도 가지고 있습니다.

또한, -d(디렉토리 존재 여부), -e(파일 존재 여부), -f(파일의 특성 확인)와 같은 다양한 파일 및 디렉토리에 관련된 기능을 제공하여 Shell Script를 작성하는 데 다양하게 쓰일 수 있습니다.

```
1 #!/bin/bash
2 if [ -z $1 ]
3 then
4     echo "What the Hell"
5 else
6     echo "$1"
7 fi
```

위의 프로그램은 앞서 배운 \$1, 즉 첫 번째 인수가 있는지를 확인하고 그것이 존재하면 그 인수를 내보내 주지만, 그것이 존재하지 않을 경우 “What the Hell”을 나타나도록 해주는 간단한 예제입니다. 아무 것도 넣지 않을 경우 다음과 같은 결과가 나오게 됩니다.

```
zed@zed-VirtualBox:~/linux132/week2$ chmod +x if.sh
zed@zed-VirtualBox:~/linux132/week2$ ./if.sh
What the Hell
zed@zed-VirtualBox:~/linux132/week2$
```

Workout

1. 파일 이름을 가지고 있는 텍스트 파일을 만든 뒤 그 파일을 생성하는 Script를 만듭니다.
2. 디렉토리 내 모든 파일을 돌면서 그것이 디렉토리인 경우 삭제하는 Script를 만듭니다.

Section 4. End

Answer

1.

```
1 #!/bin/bash
2 A=$(cat member.txt)
3 for i in $A
4 do
5     $(mkdir $i)
6 done
```

2.

```
1 #!/bin/bash
2 A=*
3 for i in $A
4 do
5     if [ -d $i ]; then
6         $(rmdir $i)
7     fi
8 done
```

Section 5.

GCC & Develop Environment

Linux 상에서 코딩을 하고, 그것을 컴파일 하는 방법은 Windows에서 Visual Studio만 쓰던 우리에게는 매우 낯선 방법입니다. 이번 강의에서는 콘솔 상에서 gcc를 이용하여 간단한 C 개발 환경을 구축하도록 하겠습니다.

Supplies



GCC

GNU(GNU is Not Unix) 프로젝트의 프리웨어(freeware) 컴파일러. 본래 C 언어용 컴파일러로 시작하였으므로 GNU C Complier의 약자였으나 2.9 버전에 이르러 C뿐만이 아니라 오브젝티브(Objective) C, 패스칼(Pascal), 에이다(Ada)와 같은 언어도 지원하였으므로 GNU Compiler Collection으로 개명하였습니다.

Lecture

콘솔 상 개발을 하기 위해서는 vim과 gcc의 설정이 필요한데, 우리는 이미 개발용 vim의 세팅을 마쳤습니다. 따라서 이번에는 컴파일러인 gcc의 사용법을 배워봅니다.

- 그 전에, 리눅스에서 파일들을 압축하고 푸는 방법에 대해 알아봅시다. 리눅스에서는 tar.bz2 또는 tar.gz라는 파일 확장자명으로 압축을 하게 됩니다.

압축을 할 때에는 다음과 같은 명령어를 사용합니다.

```
$tar -czvf 압축파일명.tar.gz 압축대상경로
```

예를 들기 위해 하나의 파일을 압축해 보도록 하겠습니다. 다음과 같이 명령어를 입력해 줍니다.

```
zed@zed-VirtualBox:~$ ls
133linux      buffer_test.txt  공개      문서      비디오  음악
Monopoly-master  linux132    다운로드  바탕화면  사진   템플릿
zed@zed-VirtualBox:~$ tar czvf test.tar.gz buffer_test.txt
buffer_test.txt
zed@zed-VirtualBox:~$ ls *.tar.gz
test.tar.gz
zed@zed-VirtualBox:~$
```

디렉토리를 확인해 보면 압축 파일이 생성되었음을 확인할 수 있습니다.
자, 이제 다시 압축을 풀어봅시다. 압축을 푸는 명령어는 다음과 같습니다.

```
$tar -xvf 압축파일명
```

역시 실행을 하면 압축이 풀립니다. 압축을 푸는 파일들을 보여주면서 압축 해제를 하게 됩니다.

```
zed@zed-VirtualBox:~$ tar xvf test.tar.gz
buffer_test.txt
zed@zed-VirtualBox:~$
```

위는 모두 tar.gz의 사용법입니다. tar.bz2 역시 하나의 압축 확장자인데, 이를 사용하고 싶을 경우 위 평령어들의 z를 j로 바꿔주면 됩니다. 다음과 같이 압축을 하고, 해제할 수 있습니다.

```
zed@zed-VirtualBox:~$ touch testfile.txt
zed@zed-VirtualBox:~$ tar -cjvf test.tar.bz2 testfile.txt
testfile.txt
zed@zed-VirtualBox:~$ tar -xvf test.tar.bz2
testfile.txt
zed@zed-VirtualBox:~$
```

이 두 명령어를 통해 우리는 이제 자유롭게 압축을 풀고, 해제할 수 있게 되었습니다. 다음에는 본격적으로 gcc에 대해 학습해보도록 합니다.

2. gcc와 관련 도구들을 설치합니다. 원래는 매우 복잡한 작업이었으나 이제는 이러한 작업들을 하나의 패키지를 설치함으로써 한 번에 끝낼 수 있게 되었습니다.

콘솔에 다음의 명령어를 입력 해주세요.

```
$sudo apt-get install build-essential
```

다음과 같이 다양한 개발 환경 패키지들이 설치되는 것을 확인할 수 있을 것입니다.

```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
Selecting previously unselected package build-essential.
build-essential 패키지를 푸는 중입니다 (.../build-essential_11.6ubuntu4_i386.deb에서)
...
Selecting previously unselected package fakeroot.
fakeroot 패키지를 푸는 중입니다 (.../fakeroot_1.18.4-2ubuntu1_i386.deb에서) ...
Selecting previously unselected package libalgorithm-diff-perl.
libalgorithm-diff-perl 패키지를 푸는 중입니다 (.../libalgorithm-diff-perl_1.19.0-2-3_all.deb에서) ...
Selecting previously unselected package libalgorithm-diff-xs-perl.
libalgorithm-diff-xs-perl 패키지를 푸는 중입니다 (.../libalgorithm-diff-xs-perl_0.04-2build3_i386.deb에서) ...
Selecting previously unselected package libalgorithm-merge-perl.
libalgorithm-merge-perl 패키지를 푸는 중입니다 (.../libalgorithm-merge-perl_0.08-2_all.deb에서)
man-db에 대한 트리거를 처리하는 중입니다 ...
dpkg-dev (1.16.10ubuntu1) 설정하는 중입니다.
build-essential (11.6ubuntu4) 설정하는 중입니다 ...
fakeroot (1.18.4-2ubuntu1) 설정하는 중입니다 ...
update-alternatives: using /usr/bin/fakeroot-sysv to provide /usr/bin/fakeroot (fakeroot) in 자동 모드
libalgorithm-diff-perl (1.19.02-3) 설정하는 중입니다.
libalgorithm-diff-xs-perl (0.04-2build3) 설정하는 중입니다 ...
libalgorithm-merge-perl (0.08-2) 설정하는 중입니다 ...
zed@zed-VirtualBox:~$
```

3. 설치한 개발 환경 도구를 가지고 컴파일을 시작해 봅시다.

먼저, vi를 이용해서 hello.c라는 이름의 간단한 Hello World 프로그램을 구현해 봅시다. (C 언어를 사용합니다.)

```
1 #include <stdio.h>
2
3 int main (void){
4     printf("hello, World! \n");
5     return 0;
6 }
```

이제 이 파일을 컴파일 해야하는데, 다음 명령어를 이용하여 컴파일할 수 있습니다.
(의미는 이후에 설명하도록 하겠습니다.)

```
$gcc -o hello hello.c
```

이 명령어를 실행하면 디렉토리에 hello라는 파일이 생성될 것입니다. 이 파일을 실행해 봅시다.
원하는 프로그램의 결과가 제대로 나왔다면 컴파일에 성공한 것입니다.

[file hello]를 입력하여 이 파일의 정보를 확인할 수도 있습니다.

```
zed@zed-VirtualBox:~$ file hello
hello: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=0x66a78a6c049d90f275
61837bb527dd334d01f3e2, not stripped
zed@zed-VirtualBox:~$
```

그러나, 방금과 같이 gcc로 컴파일을 할 때에는 그 순서와 사용법에 주의를 해야 합니다. 그렇지 않을 경우 파일이 사라지는 문제가 발생하기도 합니다.

아까 만든 파일을 가지고 이번에는 다음과 같이 입력해 줍니다.

```
$gcc -o hello.c hello
```

앞뒤가 뒤바뀐 형태입니다. 엄청나게 긴 오류가 뜨면서 컴파일이 종료됩니다.
이제 ls를 가지고 디렉토리의 파일들을 체크해 보면, 놀라운 일이 벌어집니다.

```
zed@zed-VirtualBox:~$ ls -F
133linux/      hello*      test.tar.gz  다운로드/  비디오/  템플릿/
Monopoly-master/ linux132/  testfile.txt  문서/     사진/
buffer_test.txt  test.tar.bz2  공개/      바탕화면/ 음악/
zed@zed-VirtualBox:~$
```

기존의 hello.c 파일이 사라지고 말았습니다. 컴파일 과정에서 최종 목적 파일이 hello.c로 설정이 되어있으나 이를 컴파일에 실패하면서 내부적으로 파일이 생성되지 못한 것입니다. 따라서 이전의 파일은 삭제가 되어버렸습니다.

만약에 착실하게 git을 쓰면 다음의 명령어를 가지고 잃은 파일을 복구할 수도 있습니다.

```
$git checkout hello.c
```

이렇게, 파일이 삭제되는 경우까지 초래할 수 있으므로, 컴파일을 할 때 반드시 순서에 유의하시길 바랍니다.

4. 간단한 컴파일을 해 보았으므로 이제는 gcc 자체에 대해 알아보도록 합니다.

GCC는 gnu에서 만든 프리웨어 컴파일러입니다. 더 자세한 설명은 이번 장이 시작하는 곳에도 적혀있습니다. 어느 컴파일러와 마찬가지로]전처리 → 컴파일 → 링크]의 순서로 진행이 됩니다. 프로그래밍 언어를 배우면서 배우는 내용이므로 여러분들은 이 의미에 대해 이미 알고 계시리라 생각합니다.

gcc 또한 이런 각각의 순서를 실행하는 전처리기, 컴파일러, 링커가 다음과 같이 따로 존재합니다.

전처리기 : cc1

컴파일러 : as

링커 : collect2(=:d)

그렇지만, 우리는 gcc만을 실행시켜 컴파일을 할 수 있는데, 이는 gcc가 컴파일의 전 과정을 대표하여 실행해주기 때문입니다.

5. gcc의 사용법에 대해 자세히 알아봅시다.

gcc의 기본적인 사용법은 다음과 같습니다.

```
$ gcc -g -Wall -o test file1.c file2.c file3.c
```

이는 file1.c file2.c file3.c를 컴파일한 뒤에 하나로 합쳐 출력 파일인 test를 만들으라는 소리입니다.
이 역시 c언어이기 때문에 반드시 하나의 main이 존재해야 합니다. (그렇지 않으면 실행할 수 없습니다.)

각 옵션은 다음과 같은 기능을 포함합니다.

-g : 디버깅 정보를 포함합니다.

-Wall : 모든 경고를 포함해줍니다. 이는 꼭 붙이는 습관을 들이는 것이 좋습니다.

-o : link의 의미로 c파일에 바로 이 옵션을 붙일 경우 링크의 과정까지 한번에 수행합니다.

그렇다면 한번에 링크 단계까지 수행하지 않는 방법도 있을 것입니다.

이번에는 다음과 같이 아무 명령어도 붙이지 않고 컴파일을 실행해 봅니다.

```
$ gcc hello.c
```

```
zed@zed-VirtualBox:~$ vi hello.c
zed@zed-VirtualBox:~$ gcc hello.c
zed@zed-VirtualBox:~$ ls
133linux      buffer_test.txt  linux132    testfile.txt  문서      사진
Monopoly-master  hello        test.tar.bz2  공개      바탕화면 음악
a.out          hello.c       test.tar.gz  다운로드  비디오      템플릿
zed@zed-VirtualBox:~$
```

그리고 나서 디렉토리를 확인해보면, 다음과 같이 [a.out]이라는 파일이 생성되었음을 확인할 수 있습니다.

a.out은 과거 유닉스 계통 운영 체제에서 사용하던 실행 파일과 목적 파일 형식으로, 지금은 사용하지 않습니다. 이 a.out이라는 이름은 어셈블러 출력(assembler output)을 줄인 말이라고 합니다.

a.out을 사용하던 우리가 사용하는 Linux를 포함한 대다수의 운영체제는 이후 ELF 형식으로 대체되었습니다. 현재 a.out라는 명칭은 몇몇 컴파일러나 링커에서 출력 파일명 기본값으로 사용되는 것에서 흔적을 찾을 수 있습니다.

이렇게, gcc에서도 아무 추가 명령어를 입력하지 않고 컴파일을 실행한 경우 a.out파일이 나오도록 그 흔적을 남겨 두고 있습니다.

a.out이 아닌, ELF형식으로 컴파일하기 위해서는 -c라는 옵션을 붙여주어야 합니다.

```
$ gcc -c hello.c
```

이 명령어를 수행할 경우 hello.o라는 파일이 나오게 되며, 이를 실행 파일로 만들어주는 옵션이 -o입니다.

```
$ gcc -o hello.o
```

6. 여러 파일로 나누어진 코드를 컴파일 해봅시다.

하나의 프로젝트는 여러 개의 코드(소스) 파일과 헤더 파일로 나눌 수 있습니다.

그 중, 재 사용될 수 있는 함수나 변수는 헤더 파일에 선언해 놓고, 그것을 여러 소스 파일에서 사용할 수 있습니다.

실습을 위해, [Hello, World!]를 출력하는 함수가 선언되어 있는 myprint.h와, 그것을 실행하는 main 소스 파일, 그리고 함수의 내용이 담긴 또 하나의 소스 파일을 만들어 봅시다.

우리가 배운 vi를 최대한 활용하여, 다음과 같이 코드를 구성할 수 있을 것입니다.

```

터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
1 #include <stdio.h>
2 #include "myprint.h"
3
4 int main(void)
5 {
6     printHello();
7     return 0;
8 }
~
main.c [+] 6,8-11 모두 printHello.c 1,1 모두
1 ifndef HELLO_H
2 define HELLO_H
3
4 void printHello(void);
5
6 endif
~
~
myprint.h 1,1 모두
-- 끄위넣기 --

```

파일을 모두 만든 뒤에는 컴파일 작업을 해 주어야 합니다. 앞서 컴파일 방법을 배운 것을 활용해 봅니다.

1) 전처리와 컴파일 과정으로, 어셈블리 파일을 만들어 줍니다.

```

zed@zed-VirtualBox:~$ gcc -c main.c
zed@zed-VirtualBox:~$ gcc -c printHello.c
zed@zed-VirtualBox:~$ 

```

2) 생성된 파일들을 묶어서 실행 파일로 만들어 줍니다.

```

zed@zed-VirtualBox:~$ gcc -g -Wall -o test main.o printHello.o
zed@zed-VirtualBox:~$ ls
133linux      hello.c    myprint.h    test.tar.gz  바탕화면
Monopoly-master hello.o    printHello.c  testfile.txt 비디오
a.out          linux132   printHello.o  공개          사진
buffer_test.txt main.c    test        다운로드    음악
hello          main.o    test.tar.bz2 문서        템플릿
zed@zed-VirtualBox:~$ ./test
Hello, World! zed@zed-VirtualBox:~$ 

```

이제 여러분들은 vim과 gcc를 이용하여, 프로젝트 단위로 컴파일을 할 수 있게 되었습니다.

* 본 장에서 GCC를 어느정도 익히신 분들은 다음 장으로 넘어가기 전에 Extra Section인 GDB에 대해 먼저 알아두기를 권장합니다.

Advanced

ELF 형식은 무엇인가요?

ELF는 Executable and Linkable Format의 약어로, 실행 가능하고, 링크가 가능한 포맷의 파일이라는 뜻입니다. 이 말의 키워드는 1) 실행 가능하다, 2) 링크 가능하다로 나눌 수가 있습니다. 링크에 대해서는 앞서 여러 ELF 포맷의 프로그램들을 하나로 만들어 보았기 때문에, 여기서는 실행 가능한 ELF 포맷 파일의 특징에 대해 알아보도록 하겠습니다.

실행 가능하다는 말은 프로그램이 메모리 영역에 올라가서 자원을 사용할 수 있다는 의미입니다. 이러한 것을 프로세스(Process)라고 부르는데, 프로세스에 메모리 영역을 주기 위해서 리눅스에서는 ELF 포맷을 사용하게 됩니다.

다음과 같은 명령어를 사용해 봅시다.

```
:$readelf -h /usr/bin/gcc
```

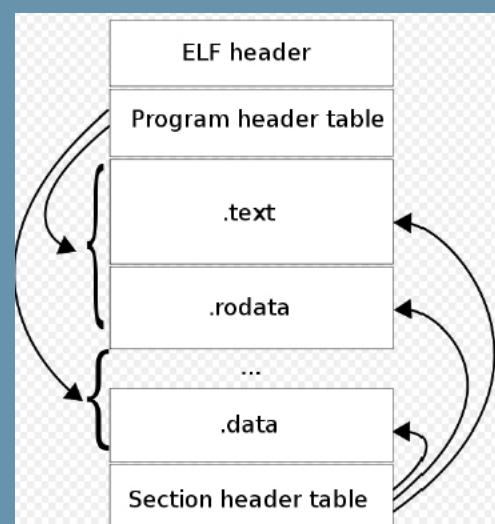
```
ELF Header:
  Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class: ELF32
  Data: 2's complement, little endian
  Version: 1 (current)
  OS/ABI: UNIX - System V
  ABI Version: 0
  Type: EXEC (Executable file)
  Machine: Intel 80386
  Version: 0x1
  Entry point address: 0x804cec4
  Start of program headers: 52 (bytes into file)
  Start of section headers: 526476 (bytes into file)
  Flags: 0x0
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 9
  Size of section headers: 40 (bytes)
  Number of section headers: 28
  Section header string table index: 27
```

readelf는 ELF포맷 파일의 정보를 확인하는 기능으로, 무언가가 많이 뜨지만 자세히는 모를 수 있습니다. 다만, 위와 같이 gcc라는 파일 역시 ELF 포맷을 따르고 있음을 확인할 수 있습니다.

이 ELF파일은 Header, Table, Section으로 구성되어 있습니다.

- 1) Header의 경우 모든 ELF포맷이 가지고 있는 것으로, 현재 ELF의 정의를 하며, Table과 Section의 위치를 결정해 주는 역할을 하고 있습니다.
- 2) Section Header Table의 경우 링크 가능 파일에 들어 있으며, 각 Section들에 대한 정의를 담당하고 있습니다.
- 3) Program Header Table의 경우 실행 가능 파일에 들어 있으며, 만약 링크를 통해 각 프로그램들을 하나의 실행 파일로 만들었다면, 이 정보들은 이 곳에 저장됩니다. 이 정보를 세그먼트(Segment)라고 합니다.

결론적으로 말하면 ELF는 섹션이나 세그먼트를 나누어 프로그램의 실행과 링킹을 원활하게 하기 위해 사용되는 파일 포맷입니다.

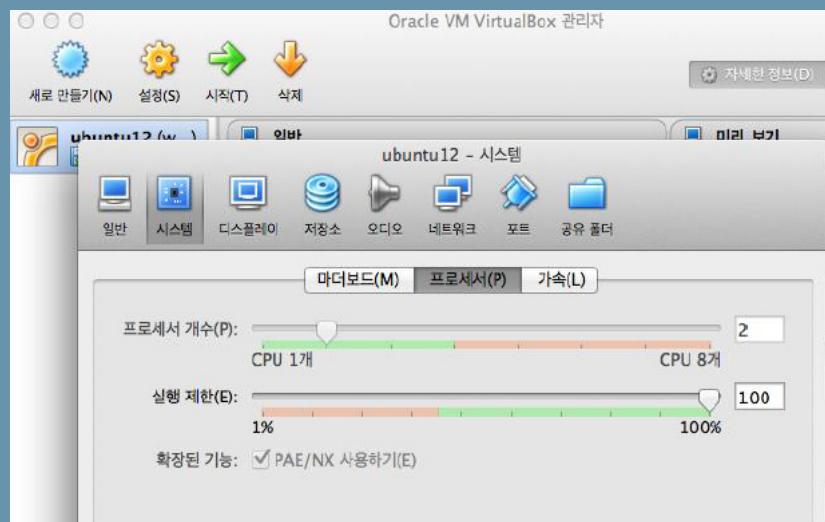


Section 6.

Make

이번 장에서는 리눅스에서 컴파일을 자동으로 수행해주는 기능인 Make에 대해 배웁니다. 기본적인 사용법과 응용 사용법 모두를 다룹니다.

Supplies



Lecture

Make에 대하여 알아봅시다.

1. Make의 기본적인 사용법에 대해 살펴봅니다.

프로젝트의 규모가 커질 경우 파일의 갯수도 그와 비례해서 늘어나기 때문에, 이것을 일일히 수동으로 컴파일하는 것은 매우 힘든 일입니다. 이러한 경우 자동으로 모든 빌드를 수행해주는 도구가 바로 Make입니다. Make는 Makefile이라는 파일에 적혀 있는 규칙 그대로 프로그램을 빌드합니다.

그러면 가장 먼저, Makefile이 어떠한 형태로 이루어져 있는지 살펴보도록 하겠습니다.

```

1 TARGET=test
2
3 SRCS = $(wildcard *.c)
4 OJBS = $(SRCS:.c=.o)
5 CFLAGS = -g -Wall
6
7 %.o: %.c
8     $(CC) $(CFLAGS) -c $<
9
10 all: $(TARGET)
11 $(TARGET): $(OJBS)
12     $(CC) -o $@ $(OJBS)
13 clean:
14     rm -rf *.o $(TARGET)

```

수업 자료를 보면 [diary.h, main.c, memo.c, calendar.c]의 네 가지 실습 파일이 있습니다.

vi로 내용을 확인해보면, 서로 엉켜있기 때문에 네 파일을 모두 한꺼번에 빌드해 주어야 하는 경우임을 알 수 있습니다.

이 파일을 이용하여 우리는 Makefile 파일을 만들 것입니다.

최종 목표는 위와 같은 형태의 파일입니다만, 지금 보면 이것이 어떤 형태인지 파악할 수 없을 것입니다.
그럼 지금부터, 가장 기초적인 makefile의 형태부터 차근차근 살펴보도록 하겠습니다.

```

1 #include "diary.h"
2
3 int main() {
4     memo();
5     calendar();
6     return 0;
7 }
~ ~
main.c
1 ifndef DIARY_H
2 define DIARY_H
3
4 include <stdio.h>
5
6 int memo();
7 int calendar();
8
9 endif
diary.h
1 include "diary.h"
2
3 int memo() {
4     printf("function %s. \n", __func__);
5 }
6
~ ~
memo.c [+]

```

가장 간단한 형태의 Makefile은 다음과 같이 생겼습니다. 아마 조금만 살펴보면 무슨 뜻인지 쉽게 파악할 수 있을 것입니다. 우리가 이전 시간에 배웠던 gcc의 내용이 그대로 담겨 있습니다. 그러면 이 파일을 vi를 통해 직접 작성해 봅시다.

```

1 all: diary
2
3 diary: memo.o calendar.o main.o
4     gcc -W -Wall -o diary memo.o calendar.o main.o
5
6 memo.o: memo.c
7     gcc -W -Wall -c memo.c
8
9 calendar.o: calendar.c
10    gcc -W -Wall -c calendar.c
11
12 main.o: main.c
13    gcc -W -Wall -c main.c
14
15 clean:
16     rm -rf *.o diary

```

그러면 이 파일을 가지고 테스트를 해 봅시다. 다음과 같은 명령어들을 입력하면 무슨 값들이 뜨는지 확인해 보도록 합니다.

```

:$make
:$make clean
:$make all
:$touch main.c
:$make
:$rm diary
:$make
:$touch diary.c
:$make

```

처음 make를 실행하면 다음과 같은 형태의 명령어가 실행되는 것을 볼 수 있습니다.

```

zed@zed-VirtualBox:~/linux132/week3/make_ex$ make
gcc -W -Wall -c memo.c
gcc -W -Wall -c calendar.c
gcc -W -Wall -c main.c
gcc -W -Wall -o diary memo.o calendar.o main.o
zed@zed-VirtualBox:~/linux132/week3/make_ex$ 

```

ls를 해보면, 이렇게 diary라는 파일이 생긴 것을 확인할 수 있습니다.

```

zed@zed-VirtualBox:~/linux132/week3/make_ex$ ls
Makefile  Makefile.4  Makefile.99  diary  main.o
Makefile.2  Makefile.5  calendar.c  diary.h  memo.c
Makefile.3  Makefile.6  calendar.o  main.c  memo.o
zed@zed-VirtualBox:~/linux132/week3/make_ex$ 

```

그러면, 다시 make를 수행해 봅시다. 그러면 다음과 같이 수행할 것이 없다는 메시지를 던져줍니다.

```

zed@zed-VirtualBox:~/linux132/week3/make_ex$ make
make: `all'를 위해 할 일이 없습니다

```

이미 diary라는 완성본이 존재하기 때문입니다. 때문에 다시 make를 수행하기 위해서는 기존의 수행했던 작업을 모두 지워줄 필요가 있습니다.

그런 경우 [make clean]을 사용하면 됩니다. 앞서 만든 Makefile에도 나와있듯이 이 명령은 rm 동작을 수행합니다.

```
zed@zed-VirtualBox:~/linux132/week3/make_ex$ make clean
rm -rf *.o diary
zed@zed-VirtualBox:~/linux132/week3/make_ex$ ls
Makefile  Makefile.3  Makefile.5  Makefile.99  diary.h  memo.c
Makefile.2  Makefile.4  Makefile.6  calendar.c  main.c
zed@zed-VirtualBox:~/linux132/week3/make_ex$
```

컴파일되었던 .o 파일과 diary 파일이 모두 사라진 것을 확인할 수 있습니다.

[make all]은 make와 같은 동작을 수행합니다. 최종 완성본인 all을 만들으라는 의미를 가지고 있습니다.

```
zed@zed-VirtualBox:~/linux132/week3/make_ex$ make all
gcc -W -Wall -c memo.c
gcc -W -Wall -c calendar.c
gcc -W -Wall -c main.c
gcc -W -Wall -o diary memo.o calendar.o main.o
zed@zed-VirtualBox:~/linux132/week3/make_ex$
```

[touch main.c]를 입력하여 main.c에 대한 변화를 일으켜 봅니다. 실제로는 아무 일도 일어나지 않았지만 Make는 이를 main.c 파일에 변화가 일어난 것으로 인식합니다.

따라서 다시 make(또는 make all)를 입력할 경우 main.c와 관련된 부분이 다시 컴파일되는 것을 확인할 수 있습니다.

```
zed@zed-VirtualBox:~/linux132/week3/make_ex$ make
gcc -W -Wall -c main.c
gcc -W -Wall -o diary memo.o calendar.o main.o
```

이번에는 diary 파일을 make clean이 아닌 rm 명령어로 지우고 make를 수행해 봅시다.

```
zed@zed-VirtualBox:~/linux132/week3/make_ex$ rm diary
zed@zed-VirtualBox:~/linux132/week3/make_ex$ make
gcc -W -Wall -o diary memo.o calendar.o main.o
zed@zed-VirtualBox:~/linux132/week3/make_ex$
```

diary 파일에 관련된 부분만 재 컴파일이 이루어집니다.

이와 같이 make는 수정 부분에 대한 것을 체크하여 그 부분을 다시 컴파일하는 것을 알 수 있습니다. make clean 명령은 모든 부분을 초기화 시킴으로써 모든 컴파일 동작이 다시 일어날 수 있도록 합니다.

이제 본격적으로 Make 기능의 구조에 대해서 알아보도록 하겠습니다.

2. Make의 구조에 대해 살펴보도록 합니다.

Make 명령어는 3개의 영역으로 구성됩니다. [타겟, 종속 항목, 명령 행]이 바로 그것입니다. 그리고, 다음과 같은 규칙을 따라야 합니다.

- 1) 종속 항목은 하나 이상 올 수 있다.
- 2) 명령어 역시 한 줄 이상 올 수 있다.
- 3) 명령어 앞에는 반드시 탭이 있어야 한다.

타겟	종속항목
9	<code>calendar.o: calendar.c</code>
명령어	
10	<code>gcc -W -Wall -c calendar.c</code>

즉, 타겟을 만들어 달라는 뜻으로, 종속 항목과 명령어를 실행해서 그 동작을 하도록 하는 것입니다. 지금까지 매운 내용을 정리하면 다음과 같습니다.

- make : 최종 완성본인 all, 즉 제일 처음 위치한 것을 만드는 명령을 수행합니다.
- make all : make와 같은 기능을 합니다.
- make clean : 종속항이 없어도 됩니다. 명령어를 수행합니다. ↴
- 명령항에는 아무 명령어나 올 수 있습니다.

기본적으로 Makefile을 사용하기 위한 과정은 모두 마쳤습니다.

그렇지만 Make는 더욱 더 편리하고 간편하게 사용할 수 있도록 여러 가지 기능을 제공하고 있는데, 고급 유저가 되기 위해서는 그 기능들에 대해서도 살펴볼 필요가 있습니다.

3. Make의 여러가지 응용 기능들에 대해 살펴봅니다. - 매크로

Make는 더욱 더 다양하게 사용하고, 또 쉽게 사용하기 위한 강력한 기능들을 제공합니다. 물론 Make를 만들어주는 여러 프로그램들이 존재하지만, 그 Makefile 파일을 이해하는 것 역시 중요하기 때문에 여기서는 추가로 다루도록 하겠습니다. (13년도 3학기에는 빠진 내용이지만 충분히 시간들 들여 따라할 수 있는 내용입니다.)

첫 번째 기능은 매크로입니다.

매크로는 기본값이 있거나 의미 있는 값들로 원래 직접 입력했던 것들을 다시 사용할 때 편리합니다. 우리가 통용적으로 알고 있는 매크로의 의미와 같습니다. 다음과 같은 매크로들을 주로 사용합니다.

CC : 컴파일러

LD : 링커

CFLAGS : 컴파일 시 옵션

LDFLAGS : 링크 시 옵션

이것을 사용하는 방법은 이전에 배웠던 Bash의 변수 사용법과 유사합니다. 설명을 하는 것보다 예제를 보는 것이 이해하기 쉬울 것입니다.

```

1 CC = gcc
2 CFLAGS = -W -Wall
3 all: diary
4
5 diary: memo.o calendar.o main.o
6   ${CC} ${CFLAGS} -o diary memo.o calendar.o main.o
7
8 memo.o: memo.c
9   ${CC} ${CFLAGS} -c memo.c
10
11 calendar.o: calendar.c
12   ${CC} ${CFLAGS} -c calendar.c
13
14 main.o: main.c
15   ${CC} ${CFLAGS} -c main.c
16
17 clean:
18   rm -rf *.o diary

```

다음과 같이 상단부에 매크로 변수에 무엇을 담을지를 입력해둡니다. (선언)

그리고 해당되는 부분에서 Bash 변수와 같이 \${매크로}와 같은 방식으로 사용합니다.

그렇지만 Bash와 같이 공백에 대한 제약은 없습니다.

여러분도 한번 따라서 바꿔보시기 바랍니다.

그런데, 매크로 중에는 파일 부분의 타겟을 설정해주는 [자동 매크로]도 존재합니다. 다음과 같은 매크로를 주로 사용합니다.

변수명	의미
\$^, \$<	모든 종속항의 리스트
\$@	현재 타겟
\${@F}	현재 타겟의 파일 부분
\${@D}	현재 타겟의 디렉토리 부분

이 자동 매크로를 가지고 다음과 같이 더욱 더 간소화시킬 수 있게 됩니다.

```

1 CC    = gcc
2 CFLAGS = -W -Wall
3 all: diary
4
5 diary: memo.o calendar.o main.o
6     ${CC} ${CFLAGS} -o $@ $^
7
8 memo.o: memo.c
9     ${CC} ${CFLAGS} -c $^
10
11 calendar.o: calendar.c
12     ${CC} ${CFLAGS} -c $^
13
14 main.o: main.c
15     ${CC} ${CFLAGS} -c $^
16
17 clean:
18     rm -rf *.o diary

```

4. Make의 여러가지 응용 기능들에 대해 살펴봅니다. - 확장자 규칙

확장자 규칙이라 함은, 우리가 배웠던 와일드카드와 비슷합니다. %를 이용해서 모든 o와 모든 c를 표시할 수가 있는 것입니다. 예를 들어,

```
:%.o : x1 x2 x3
      yyyyyy
```

와 같이 입력한다면, 이는 모든 .o파일을 타겟으로 지정하고 명령을 반복적으로 수행하라는 뜻입니다. 자, 이것을 사용하면 이제 다음과 같이 더 간단한 Makefile을 만들 수 있습니다.

```

1 CC    = gcc
2 CFLAGS = -W -Wall
3 all: diary
4
5 diary: memo.o calendar.o main.o
6     ${CC} ${CFLAGS} -o $@ $^
7
8 %.o: %.c
9     ${CC} ${CFLAGS} -c $<
10
11 clean:
12     rm -rf *.o diary

```

사실 이 외에도 기본 확장자 규칙이라는 것이 있는데, 다음 명령어로 규칙들을 확인할 수 있습니다.

```
:$ make -p | more
```

사실 이것을 보면 지금 배운 %.o를 해주지 않아도 된다는 놀라운 사실을 알 수 있습니다.
즉, 이렇게만 해도 make가 알아서 일을 처리해 준다는 것입니다.

```

1 CC = gcc
2 CFLAGS = -W -Wall
3 all: diary
4
5 diary: memo.o calendar.o main.o
6   ${CC} ${CFLAGS} -o $@ $^
7
8 clean:
9   rm -rf *.o diary

```

5. Make의 여러가지 응용 기능들에 대해 살펴봅니다. - 와일드카드

매크로에 *.c 같은 걸 넣으면 더 편하겠죠? 앞서 %는 일일히 명령을 수행하는 명령문을 구성해야 했었지만, 와일드카드 한 문장을 넣으면 스스로 변환시켜준다면 매우 좋을 것입니다. Make는 그와 같은 기능도 제공합니다. 다음과 같이 입력하면 와일드카드 문장을 수행하게 됩니다.

```

SRCS = $(wildcard *.c)
OBJS = $(SRCS:.c=.o)

```

이는 SRCS에 모든 *.c를 받으라는 뜻이고, 아래 OBJS에 모든 *.c를 모든 *.o로 변환시켜서 넣으라는 작업을 의미합니다. 이것을 활용하면 훨씬 줄어든 결과가 나올 수 있을 것입니다.

```

1 CC = gcc
2 CFLAGS = -W -Wall
3 SRCS = $(wildcard *.c)
4 OBJS = $(SRCS:.c=.o)
5
6 all: diary
7   @echo "빌드 성공"
8
9 diary: $(OBJS)
10  ${CC} ${CFLAGS} -o $@ $^
11
12 clean:
13   rm -rf *.o diary

```

다음과 같이 줄일 수 있는데, 아까 우리가 보았던 처음의 Makefile과 같습니다.
지금까지의 내용을 잘 따라왔다면 이 문법을 충분히 이해하실 수 있을 것입니다.

Advanced

소스가 더 많아지고 라이브러리를 이것저것 활용하다보면, Makefile을 직접 만드는 데에도 한계가 오게 됩니다.
때문에 자동으로 Makefile을 만들어주는 autoconf라는 프로그램도 있습니다.
또한, 유사품으로 멀티플랫폼 빌드를 해주는 cmake도 있습니다.

여기서는 여러분들이 추가로 학습하실 수 있도록 과제로 남겨두도록 하겠습니다.

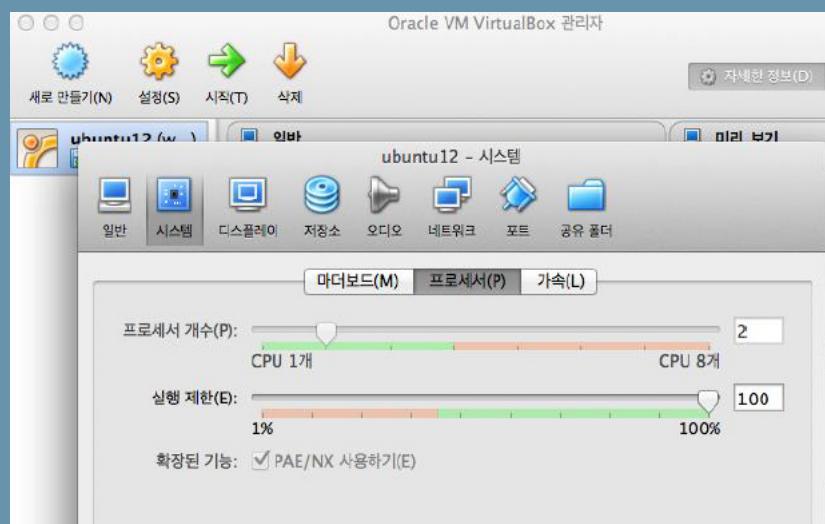
Section 6. End

Section 7.

Process & Thread

프로세스와 스레드의 개념에 대해서는 운영체제 시간에 다루므로 간략하게 넘어가며, 본 강에서는 이러한 스레드와 프로세스가 리눅스에서 어떻게 작동하는지를 알아보는 시간을 가지도록 하겠습니다.

Supplies



이번 강에서는 Mult-Threading에 대한 실습이 있으므로, VirtualBox의 CPU를 최소 2개로 바꾸어주어야 합니다.

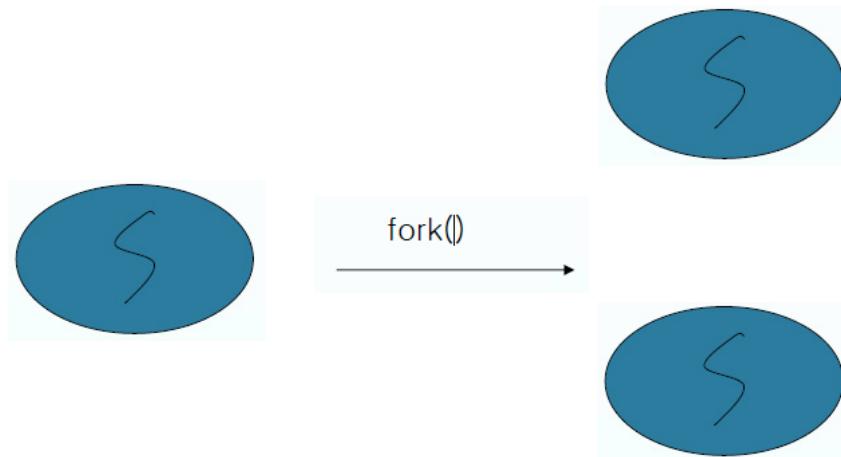
Lecture

프로세스에 대해 간략히 알아봅시다.

1. fork()는 하나의 프로세스를 복사하는 역할을 하는 시스템 콜입니다.

fork() 명령어를 이용하지 않고는 절대로 다른 프로세스를 만들 수 없습니다.

프로세스는 그 자체가 독립된 기능을 하므로 복제가 된 이후에는 다른 프로세스 ID(pid)와 메모리 영역을 가지게 됩니다. 따라서 원래 동작하는 프로세스와 복제된 프로세스, 이렇게 완전히 독립된 두 개의 프로세스가 작동할 수 있게 됩니다.



프로세스가 복제가 되면 원래의 프로세스를 복제된 프로세스의 **부모 프로세스**라고 부르며, 복제된 프로세스를 원래 프로세스의 **자식 프로세스**라고 말합니다.

실제로 fork()를 실행할 경우 return값을 가지게 되어 이를 통해 부모와 자식을 구분할 수 있습니다.

부모 프로세스는 자식 프로세스의 pid를 반환하며, 자식 프로세스는 0을 반환합니다.

(자식 프로세스의 pid가 0이라는 의미는 아니며, 이것이 자식 프로세스로 생성되었음을 의미하는 것입니다.)

이러한 성질을 알고 다음의 예를 살펴보도록 합시다.

```

터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main() {
5
6     pid_t pid;
7     int status;
8
9     printf("Hello? \n");
10
11    pid = fork();
12    printf("pid = %d \n", pid); // show pid
13    if(pid !=0){
14        // parent
15        wait(&status);
16        printf( Parent: BYE, %d. \n", pid);
17    } else {
18        //child
19        int i = 0;
20        while(i< 10) {
21            sleep(1);
22            printf("%d \n", i++);
23        }
24
25        printf("Child: BYE, %d. \n", pid);
26    }
27
28    return 0;
29 }

test.c [+] 15,8-14 모두

```

위와 같은 프로그램을 작성한 후, 실행해 봅시다.

```

zed@zed-VirtualBox:~$ ./test
Hello?
pid = 3853
pid = 0
0
1
2
3
4
5
6
7
8
9
Child: BYE, 0.
Parent: BYE, 3853.
zed@zed-VirtualBox:~$ 

```

다음과 같이 부모 프로세스가 자식 프로세스를 복사한 뒤,
자식 프로세스가 10초간 자신의 일을 실행하는 것을 기다리고,
그것이 모두 종료되었을 경우 부모 프로세스를 마치는 코드입니다.

간단한 수준이니 조금 살펴보면 이해할 수 있을 것입니다.

그렇다면 여기서 의문점이 드는 것이, 이러한 fork()는 무슨 용도로 사용하나
는 것일겁니다. 실제로 fork()는 자기 자신을 복사하는 역할만 합니다. 이것
만 가지고는 큰 의미를 가지지 않습니다.

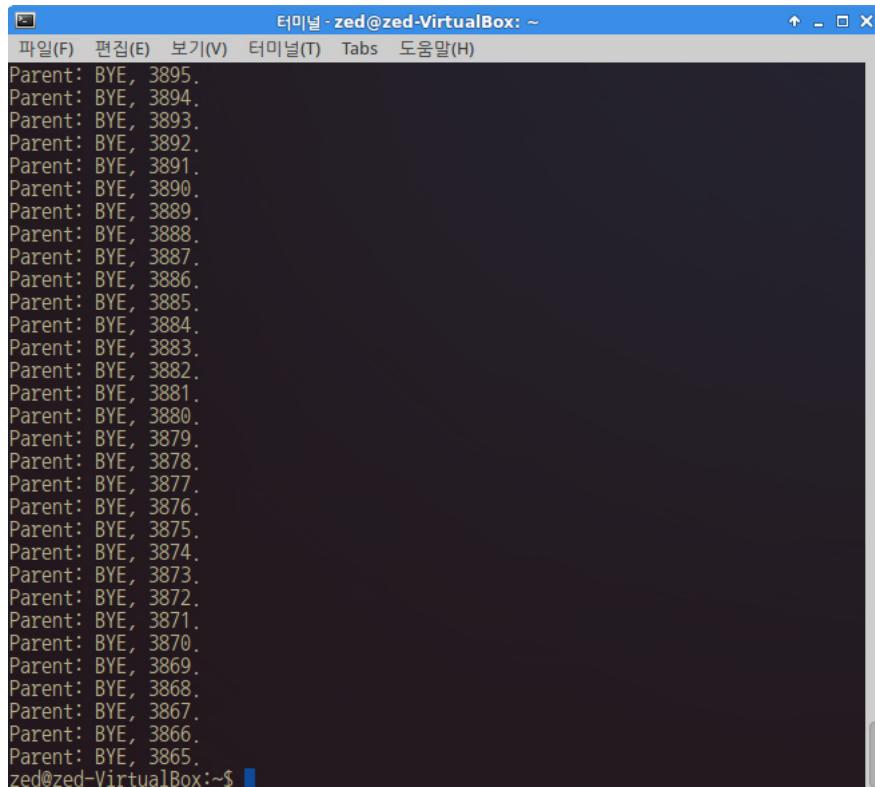
이 때 프로세스를 바꾸어 실행시켜주는 명령어가 바로 exec()입니다.
exec()에 대해서는 위의 예제 코드를 살짝 바꾼 다음의 코드를 보며 살펴보
도록 하겠습니다.

2. exec()는 실행하고 있는 프로세스를 바꾸어줍니다.

이전의 코드에서 Child 부분의 코드를 다음과 같이 변경하였습니다. 이 프로그램을 실행해봅시다.

```
    } else {
        //child
        execl("./test","./test","5",NULL);
        printf("can you see me?\n");
    }

    return 0;
}
```



The terminal window shows a series of "Parent: BYE, <process_id>" messages, where <process_id> is a sequence of numbers starting from 3895 and decreasing down to 3865. This indicates that the program is creating many child processes and switching between them rapidly.

다음과 같이 하나의 자식 프로세스가 다시 부모 프로세스가 되어 자식 프로세스를 생성하는 작업을 무수히 거친 뒤에 하나씩 종료되는 과정을 거치고 있습니다. 이렇게 프로세스를 생성하고, 다시 교체하고, 원래의 프로세스로 돌아오는 과정은 굉장히 많은 오버헤드가 발생합니다.

이러한 경우에 대한 해결책으로 스레드를 이용합니다.

스레드는 프로세스 복제와 달리 하나의 프로세스 속에서 서로 코드, 데이터, 힙을 공유하고 있으며 이들을 함수 단위로 병렬 실행할 수 있게 해 줍니다.

스레드에 대해서는 다음 강에서 자세히 알아보겠습니다.

Lecture

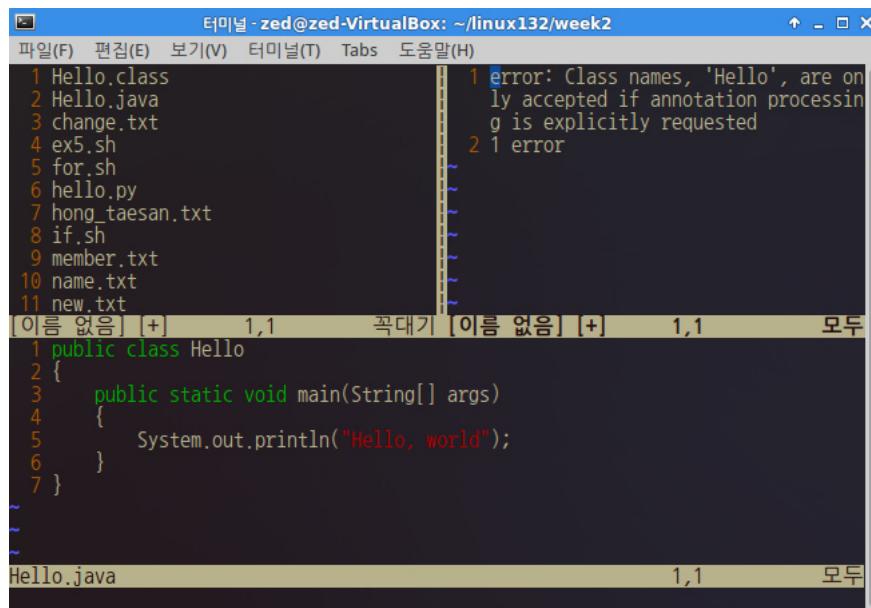
스레드에 대해서 알아봅시다.

1. 리눅스의 작업 관리 단위는 태스크(Task)입니다.

운영체제 시간에 배운 커널 스레드(Kernel Thread)의 역할을 하는 작업 단위를 리눅스에서는 태스크(Task)라고 부릅니다. 말 그대로, 시스템과 연결하여 기능을 수행하는 스레드를 말하며, 사용자는 하나의 프로세스나 스레드를 하나의 Task에 연결시켜서 작업을 수행할 수 있습니다.

스레드(Thread)란, **프로그램의 흐름, 또는 그 흐름의 단위**라는 정의를 가지고 있습니다.
지금까지 여러가지 프로그램을 코딩해오면서 무언가 흐름이 있다는 것을 느끼셨다면,
그것이 바로 하나의 스레드의 개념입니다. 실제로 CPU에서 실행되는 하나의 실행 단위가 바로 스레드입니다.

우리가 지금까지 짜 왔던, 그리고 앞으로도 수 없이 짤 많은 프로그램들은 싱글 스레디드 프로그램입니다.



The screenshot shows a terminal window titled "터미널 - zed@zed-VirtualBox: ~/linux132/week2". The window lists several files in the current directory: Hello.class, Hello.java, change.txt, ex5.sh, for.sh, hello.py, hong_taesang.txt, if.sh, member.txt, name.txt, and new.txt. Below this, the Java code for "Hello.java" is displayed:

```

1 public class Hello
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello, world");
6     }
7 }
```

On the right side of the terminal, there is an error message from the Java compiler:

```

1 error: Class names, 'Hello', are only accepted if annotation processing is explicitly requested
2 1 error
```

이런 간단한 프로그램부터, Lab 수업에서 만드는 복잡한 프로그램까지 싱글 스레드 프로그램을 만든 경우가 굉장히 많았을 것입니다. 그렇지만 모든 프로그램이 하나의 흐름만을 가지고 동작하지는 않습니다.

하나의 프로세스에서 두 가지 이상의 동작을 동시에 수행할 수도 있는데, 이러한 동작을 하기 위해서는 하나 이상의 흐름이 더 필요할 것입니다. 이러한 것을 **멀티 스레디드 프로그램(Multi-Threaded Program)**이라고 합니다.

이번 시간에 우리는 이러한 멀티 스레드 프로그램을 만들어 볼 것입니다. 그것을 위해서 Java 언어를 사용합니다.

2. Java로 스레드를 만들어 봅시다.

Java에서 스레드를 만드는 방법은 크게 다음의 두 가지로 나뉩니다.

(1) Thread 클래스 상속

(2) Runnable 인터페이스 Implements

Java를 배우면 알 수 있지만, Java 언어의 경우 다중 상속이 불가능하기 때문에 Runnable 인터페이스를 더 많이 사용합니다. 따라서 이번 시간에는 (2)번의 Runnable 인터페이스를 가지고 면티 스레드를 만들어 볼 것입니다. 이 인터페이스를 추가하기 위해서는 한 가지 조건이 있는데, 바로 public void run() 메소드를 구현해야 한다는 것입니다.

```
1 public class Test1 implements Runnable {
2     public void run() {
3         System.out.println("thread");
4     }
}
```

이 run() 메소드는 하나의 스레드가 추가될 때, 그 스레드가 실행할 부분을 나타내는 것입니다.

다시 말하면, 처음 실행하는 스레드는 싱글 스레드 프로그램과 같이 main() 메소드를 실행하지만, 만약 이 main 메소드가 실행되면서 하나 이상의 스레드를 더 요구할 경우, 그 스레드는 run() 메소드를 실행하게 되는 것입니다.

```
1 public class Test implements Runnable {
2     public void run() {
3         long tid = Thread.currentThread().getId();
4         try {
5             Thread.sleep(1000);
6         } catch (InterruptedException e) {
7             System.out.println(e.getMessage());
8         }
9         System.out.printf("%d thread test \n", tid);
10    }
11
12    public static void main(String[] args) {
13        long mid = Thread.currentThread().getId();
14        System.out.println("Main Start");
15        Test test = new Test();
16        Thread thread = new Thread(test);
17        thread.start();
18        System.out.printf("%d Main end \n", mid);
19    }
20}
21
```

vi로 다음과 같은 프로그램을 작성한 후에 실행해 봅시다.

javac로 컴파일할 경우 java라는 확장자까지 붙여주어야 하지만, java로 실행할 경우에는 클래스의 이름만 써 주어야 합니다.

```
zed@zed-VirtualBox:~$ javac Test.java
zed@zed-VirtualBox:~$ java Test
Main Start
1 Main end
7 thread test
zed@zed-VirtualBox:~$
```

Main이 순차적으로 종료되었음에도 프로그램에서 실행되는 것이 남아있는데, 이 것이 바로 run() 메소드를 도는 두 번째 스레드입니다.

Java의 스레드는 다음과 같이 동작합니다.

- 1) Java 프로그램을 시작하면, JVM이 스레드를 하나 만듭니다. 이 스레드는 main() 메소드를 실행합니다.
- 2) 자신을 인자로 받는 Thread를 생성할 수 있습니다. 이 스레드를 실행하려면 start() 메소드를 이용합니다.
- 위의 예제 프로그램의 다음 부분입니다.

```
Test test = new Test();
Thread thread = new Thread(test);
thread.start();
```

또한, 이 스레드들은 다음과 같은 특징을 가지고 있습니다.

1) 코드와 힙 영역을 공유합니다.

- 클래스 멤버변수, static으로 되어있는 변수 등은 모두 공유할 수 있습니다.

2) 스택, 레지스터는 개별적으로 사용합니다.

- 메소드 내의 지역 변수는 개별적으로 가지고 공유하지 않습니다.

위와 같은 스레드의 특징을 더 자세히 알아보기 위해 스레드를 두 개 더 만들어보도록 하겠습니다.

다음과 같은 프로그램을 작성해보고, 실행해보도록 하겠습니다.

```
1 public class Test2 implements Runnable {
2     int m = 0; //class variable in heap
3     public void run() {
4         int l = 0; //local variable in stack
5         l++;
6         long tid = Thread.currentThread().getId();
7         //try block is needed for Thread.sleep
8         try {
9             //sleep for 1 second
10            Thread.sleep(1000);
11        } catch (InterruptedException e) {
12            System.out.println(e.getMessage());
13        }
14        m++;
15        System.out.printf("Tid %d thread l:%d m:%d\n",
16                           tid, l, m);
17    }
18
19    public static void main(String[] arg) {
20        long mid = Thread.currentThread().getId();
21        System.out.println("Main start");
22        Test2 test = new Test2();
23        Thread thread1 = new Thread(test);
24        Thread thread2 = new Thread(test);
25        thread1.start();
26        thread2.start();
27        System.out.printf("Tid %d Main end m: %d\n", mid, test.m);
28    }
29 }
```

Test2.java 1,1
"Test2.java" 29L, 772C

1) 클래스 변수, 지역 변수가 각각 어떻게 변하는지 관찰해 봅시다.

- 클래스 변수 m은 공유하지만, 지역 변수 l은 자체 처리합니다.

2) 메인 스레드와 두 추가 스레드의 생명주기를 관찰해 봅시다.

- 메인, 스레드1, 스레드2의 만들어진 순서대로 종료하는 것을 확인할 수 있습니다. 특히 run()에 sleep을 걸어놓았기 때문에 main이 가장 먼저 끝나고, 1초 뒤에 두 개의 스레드가 순차적으로 종료됨을 알 수 있습니다.

```
Main start
Tid 1 Main end m: 0
Tid 7 thread l:1 m:1
Tid 8 thread l:1 m:2
zed@zed-VirtualBox:~/linux132/week5/thread$
```

3. join()으로 다른 스레드를 기다릴 수 있습니다.

join() 메소드를 사용하면 해당 메소드가 완료 될 때까지 join() 메소드를 부른 스레드는 일시 중지됩니다. 이전의 코드에 다음과 같은 join() 메소드를 추가하고 실행해 봅시다.

```

19  public static void main(String[] args)
20      throws InterruptedException {
21      long mid = Thread.currentThread().getId();
22      System.out.println("Main start");
23      Test2 test = new Test2();
24      Thread thread1 = new Thread(test);
25      Thread thread2 = new Thread(test);
26      thread1.start();
27      thread2.start();
28      thread1.join();
29      thread2.join();
30      System.out.printf("Tid %d Main end m: %d\n", mid, test.m);
31  }
32 }
```

다음과 같은 결과가 나타나는 것을 확인할 수 있습니다.

```

zed@zed-VirtualBox:~/linux132/week5/thread$ java Test2
Main start
Tid 7 thread l:1 m:1
Tid 8 thread l:1 m:2
Tid 1 Main end m: 2
zed@zed-VirtualBox:~/linux132/week5/thread$
```

main() 메소드에서 join()을 걸었기 때문에 이전과 다르게 main()이 기존의 스레드들이 모두 끝나기를 기다린 뒤에 종료되는 것을 알 수 있습니다. 이렇게 join()을 활용하면 스레드들간의 종료 시점을 원하는 대로 조절할 수 있습니다.

이러한 join을 이용해서 다음과 같은 프로그램을 작성해 봅시다.

[1부터 20까지 출력하기]

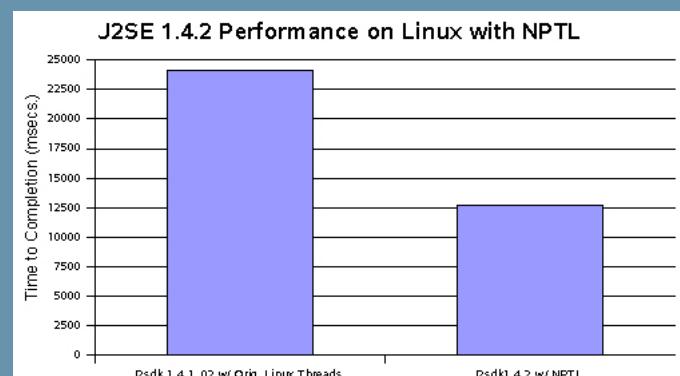
- 스레드 4개로 만듭니다.
- tid와 숫자를 한 번씩 출력하고 0.1초씩 쉭니다.
- 메인 스레드로 최종값을 한 번 더 찍습니다.
- 클래스 변수와 join()을 사용합니다.

Advanced

리눅스에서는 프로세스, 스레드와 커널 스레드가 1:1로 대응하는 NPTL 방식을 사용하고 있습니다.

스레드의 1:1 대응과 M:N 대응이 갖는 장점과 단점은 운영체제에서 배운 내용과 다르지 않지만, 리눅스에서는 스케줄링의 오버헤드가 O(1)인 방식을 고안함으로써, 1:1 대응 방식인 NPTL을 사용합니다.

<http://en.wikipedia.org/wiki/NPTL>



우리가 배운 내용을 활용한다면, 다음과 같은 프로그램을 만들 수 있습니다.

```

1 public class Test5 implements Runnable {
2     static final int MAX = 20;
3     static final int NT = 4; //number of thread
4     int m = 0; //class variable in heap
5     public void run() {
6         long tid = Thread.currentThread().getId();
7         while (m < MAX) {
8             m++;
9             System.out.printf("Tid %d value: %d\n", tid, m);
10            try {
11                //sleep for 1 second
12                Thread.sleep(100);
13            } catch (InterruptedException e) {
14                System.out.println(e.getMessage());
15            }
16        }
17    }
18
19    public static void main(String[] args)
20        throws InterruptedException {
21        Test5 test = new Test5();
22        Thread[] at = new Thread[NT];
23        for(int i = 0; i < NT; i++) {
24            at[i] = new Thread(test);
25            at[i].start();
26        }
27        for(Thread t:at)
28            t.join();
29        System.out.println(test.m);
30    }
31 }
```

Test5.java 1,1 모두

```

Tid 7 value: 1
Tid 10 value: 4
Tid 9 value: 3
Tid 8 value: 2
Tid 10 value: 5
Tid 9 value: 6
Tid 8 value: 7
Tid 7 value: 8
Tid 8 value: 9
Tid 9 value: 10
Tid 10 value: 11
Tid 7 value: 12
Tid 9 value: 13
Tid 8 value: 14
Tid 10 value: 15
Tid 7 value: 16
Tid 8 value: 17
Tid 9 value: 18
Tid 10 value: 19
Tid 7 value: 20
20
```

그렇지만 결과는 의도한 것과 많이 다릅니다. 이는 한 메모리에 동시에 값을 쓰거나, 동시에 읽는 일들이 발생하기 때문입니다.

이렇게, 이러한 join()만으로 스레드들의 관계가 완전히 해결되지는 않습니다. 우리에게는 아직 deadlock과 synchronizing(동기화)의 문제가 남아있는데,

이는 다음 강에서 알아보도록 하겠습니다.

Lecture

동기화와 그로 인해 일어날 수 있는 데드락 현상에 대해 알아봅니다.

1. 동기화는 정확한 결과를 보장해줍니다.

동기화란 CPU 개수에 관계 없이 한 번에 하나의 실행만을 보장하는 것을 말합니다. 즉, 여러 스레드들이 동시에 실행하지 않도록 하여 변수가 정확하기 않게 나타나는 현상을 막을 수 있는 것입니다. 특히 우리가 사용하는 언어인 java에서는 메소드 단위로 동기화하는 방법과, 클래스 단위로 동기화하는 방법 두 가지를 제공하고 있습니다.

그렇지만, 아무래도 하나의 스레드만을 실행하도록 하는 방법이기 때문에 지나치게 많은 동기화를 적용하는 것은 성능 저하를 유발할 수 있습니다. 때문에 꼭 필요한 곳에만 동기화를 적용하는 것이 필요합니다.

첫 번째로 알아볼 동기화 방법은 메소드 단위 동기화입니다.

하나의 메소드 앞에 synchronized를 붙이면 메소드 단위로 동기화가 되며, 해당 메소드 안에서 공유되는 변수들에 대해서는 동기화를 보장해줍니다. 이 말은, 공유가 되지 않는 변수들은 여전히 그 특성을 가지며 동기화가 되지 않음을 의미하니 여기에 주의해서 사용해야 합니다.

```

7     synchronized void sum() {
8         n++;
9         if( n <= max)
10        {
11            sum += n;

```

위의 코드에서 변수 n은 메소드 밖에 있는 공유 변수이기 때문에, synchronized가 붙은 경우 동시에 접근하지 못하고 한번에 하나의 메소드가 실행될 때 그 메소드만이 해당 변수에 접근할 수 있으며, 다른 메소드들은 대기하게 됩니다.

일전에 만든 1부터 20까지 세는 프로그램을 다음과 같이 수정할 수 있습니다.

```

public synchronized void increase(long id)
{
    while (m < MAX) {
        m++;
        System.out.printf("Tid %d value: %d\n", id, m);
        try {
            //sleep for .1 second
            Thread.sleep(100);
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

이와 같이 동기화를 적용한다면, 결과는 다음과 같이 제대로 나오게 됩니다.

```
zed@zed-VirtualBox:~/linux132/week6$ java Test6
Tid 7 value: 1
Tid 7 value: 2
Tid 7 value: 3
Tid 7 value: 4
Tid 7 value: 5
Tid 7 value: 6
Tid 7 value: 7
Tid 7 value: 8
Tid 7 value: 9
Tid 7 value: 10
Tid 7 value: 11
Tid 7 value: 12
Tid 7 value: 13
Tid 7 value: 14
Tid 7 value: 15
Tid 7 value: 16
Tid 7 value: 17
Tid 7 value: 18
Tid 7 value: 19
Tid 7 value: 20
20
zed@zed-VirtualBox:~/linux132/week6$
```

이와 같이 하나의 메소드 단위로 동기화를 적용할 수도 있지만, 클래스 단위로 동기화를 적용하는 방법도 있는데, 이를 **클래스 통기화, 또는 동기화 블록**이라고 부릅니다.

```
7     void sum() {
8         synchronized(data) {
9             data.n++;
10            if( n <= max)
11            {
12                data.sum += data.n;
13            }
14        }
15    }
```

원래 data라는 변수는 공유가 되지 않는 클래스 변수이지만, 다음과 같이 메소드 안에 블록을 형성할 경우 해당 블록 내에서는 data라는 변수 역시 동기화 설정이 될 수 있게 됩니다. 이와 같이 클래스 단위 변수에서 동기화를 해야 할 경우가 발생할 시에 동기화 블록을 만들어줍니다.

2. 동기화를 적용하다보면 데드락 현상이 유발될 수 있습니다.

데드락(교착) 현상이란 두 개 이상의 작업이 서로 맞물려 상대방의 작업이 끝나기를 기다리는 현상으로, 결과적으로 아무것도 완료되지 못한 채로 가만히 멈추게 되는 상태를 말합니다. 이와 같은 현상은 멀티스레드, 또는 멀티프로세스 시스템에서 흔히 발견될 수 있습니다.

이러한 데드락이 발생하기 위한 조건은 흔히 다음의 네 가지로 나뉩니다.

Workout

다음 코드의 교착 현상을 해결해봅시다.

코드 전체는 다음 장에 있으며, 해당 부분에서 교착 현상이 발생하는 두 캐릭터 간 전투 프로그램입니다.

Answer

```
63     Player first, second;
64     if(from.mPlayerID < to.mPlayerID) {
65         first = from;
66         second = to;
67     } else {
68         first = to;
69         second = from;
70     }
71     synchronized(first) {
72         synchronized(second) {
73             to.mHP -= 3000;
74             from.mHP += 3000;
75             try {
76                 Thread.sleep(100);
77             } catch (InterruptedException e) {
78                 System.out.println(e.getMessage());
79             }
80         }
81     }
82     System.out.printf("E N D: from #%d(%d) to #%d(%d)\n",
83                       from.mPlayerID, from.mHP,
84                       to.mPlayerID, to.mHP);
85 }
86 }
```

Section 8.

File System

우리가 알고 있는 물리적 저장장치에 데이터를 저장하기 위한 계층을 파일 시스템이라고 말하며, 파일에 대한 일관적이고 논리적인 관점을 제공합니다.

Supplies

Lecture

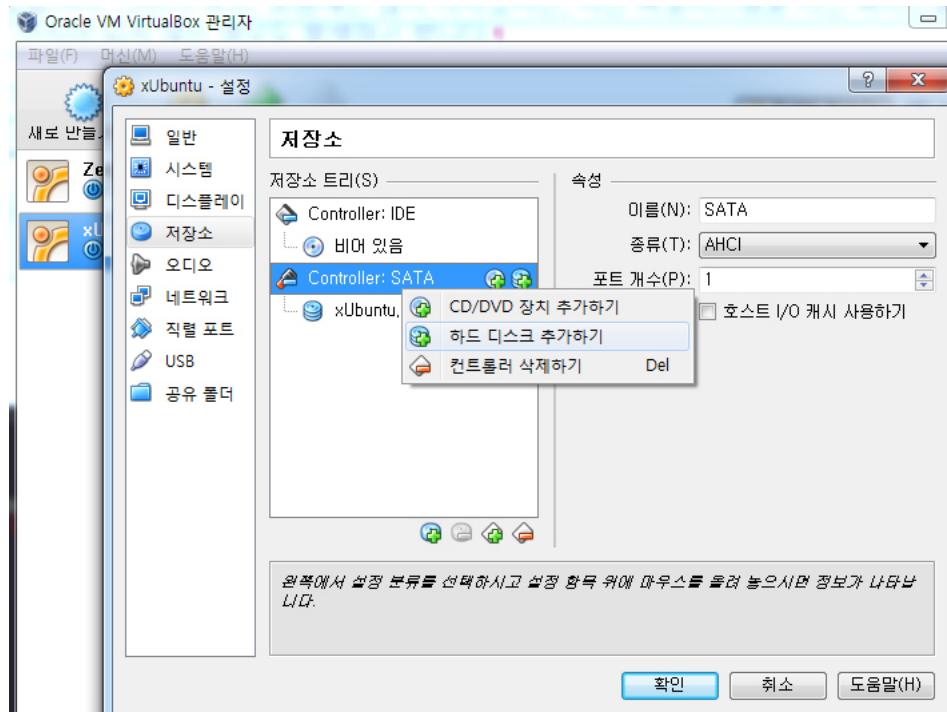
마운트와 파티션 작업, 그리고 포맷에 대해 알아봅시다.

1. 파일 시스템에 대한 간략한 소개

우리가 알고 있는 하드디스크, USB, CD-ROM 등은 모두 저장장치로 불리며, 이 안에 우리는 데이터를 물리적인 방법으로 저장하게 되고, 읽을 수 있게 됩니다. 파일시스템은 이러한 물리적 저장장치에 어떠한 방법으로 데이터를 저장하느냐, 그리고 어떻게 효율적인 공간활용을 하느냐에 대한 논리적인 부분을 담당하는 것을 말합니다. 우리는 이러한 파일시스템의 도움으로 종류와 특성이 완전히 다른 물리적 저장장치들을 동일하게 파일을 저장하고 관리하는 형태로 인식할 수 있습니다. 그런가하면, 요즘에는 많이 나아졌지만, 다른 파일시스템을 가진 파티션들, 또는 저장장치들간에 제대로 인식이 되지 않는 경우도 발생하곤 합니다.

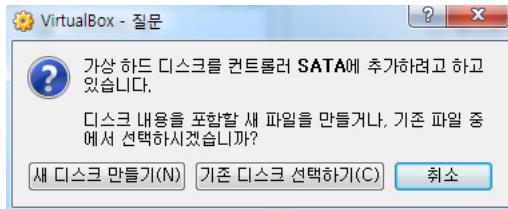
마운트(Mount)는 바로 이러한 물리적 저장장치와 논리적인 접근 지점을 연결하는 것을 말합니다. 윈도우의 C:와 D:가 하드디스크를 의미하는 것과 같습니다. 우리가 배우고 있는 리눅스의 경우 메인 드라이브가 '/(root)'에 마운트됩니다.

우리는 이 마운트(Mount)과정과 새롭게 파티션을 나누고 파일 시스템을 부여하는 실습을 진행합니다. 그러기 위해서 먼저 버추얼 머신의 가상 디스크를 추가로 생성해주어야 합니다. 이를 위해 시동되어있는 가상 머신을 종료하고, [설정]에 들어가 디스크를 추가로 생성해주세요.

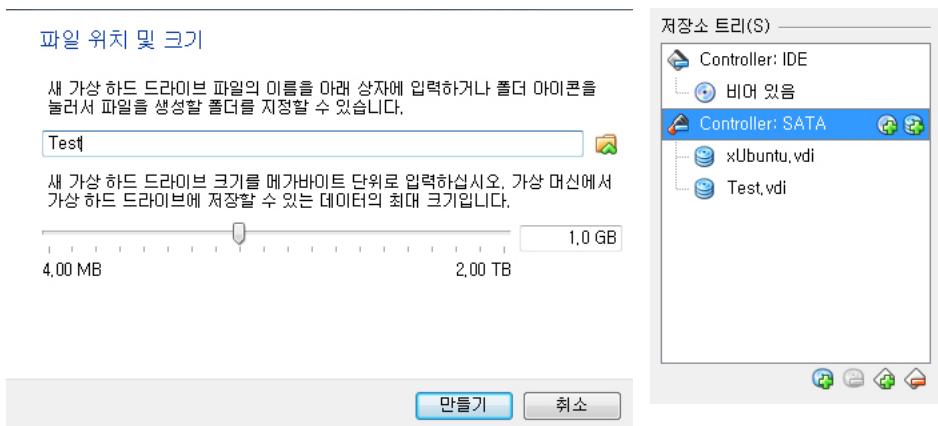


우리는 IDE가 아닌 기존에 만들어졌던 SATA 디스크에 하나를 더 추가하여 줍니다.

그렇게 되면, 다음과 같은 경고문이 뜨게 됩니다. [새 디스크 만들기]를 눌러줍니다.



이 다음 과정은 우리가 처음에 가상 디스크를 추가하던 방법과 같습니다. 자세한 내용은 1장 4페이지를 참고하면 되며, 실험용 디스크이므로 용량은 적당히 낮게 설정해주면 됩니다.



위와 같이 저장소 트리에 새로운 디스크가 생성되었다면 실습의 준비가 모두 끝난 것입니다. 이제 가상 머신을 다시 시동시킵니다.

2. 새로운 디스크에 파티션 작업을 수행해봅니다.

모든 하드웨어는 /dev 디렉토리에 디바이스 파일이라는 특수파일로 존재합니다. 다음과 같은 명령어로 우리가 만든 새로운 디스크가 인식되었음을 확인할 수 있습니다.

```
:$ ls /dev/sd* -;
```

```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
zed@zed-VirtualBox:~$ ls /dev/sd* -l
brw-rw---- 1 root disk 8,  0 1월  5 20:45 /dev/sda
brw-rw---- 1 root disk 8,  1 1월  5 20:45 /dev/sda1
brw-rw---- 1 root disk 8,  2 1월  5 20:45 /dev/sda2
brw-rw---- 1 root disk 8,  5 1월  5 20:45 /dev/sda5
brw-rw---- 1 root disk 8, 16 1월  5 20:45 /dev/sdb
zed@zed-VirtualBox:~$
```

위와 같이 /dev/sdb가 새롭게 추가되었음을 알 수 있습니다. sda는 기존의 드라이브로, 이 드라이브를 세 조각(파티션)으로 나누어 사용하고 있다는 의미입니다. 반면에 sdb는 아직 나누어진 조각, 즉 파티션이 없습니다. 디스크를 사용하기 위해서는 파티션을 생성하여 파일 시스템을 부여해주어야 합니다.

다음과 같은 명령어로 파티션 관리자를 실행할 수 있습니다.

```
:$ sudo fdisk /dev/sdb
```

```
zed@zed-VirtualBox:~$ sudo fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0x0fcab3eb.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help):
```

명령어를 입력하라는 단순한 프롬프트가 뜹니다. 여기에 우리는 다음과 같은 명령어를 순차적으로 실행할 것입니다.

- 1) p : 파티션 정보 출력(print)
- 2) n : 새로운 파티션 만들기(new)
- 3) w : 실제 디스크에 저장하기 (write)

1) p : 파티션 정보 출력(print)

p를 입력하면, 다음과 같이 파티션 정보가 뜹니다. 지금은 아무 파티션도 존재하지 않기 때문에 파티션 정보가 뜨지 않습니다.

```
Command (m for help): p
Disk /dev/sdb: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0fcab3eb

Device Boot      Start         End      Blocks   Id  System

```

2) n : 새로운 파티션 만들기(new)

새로운 파티션을 만드는 과정으로, 다음과 같은 순서로 진행됩니다.

```
Command (m for help): n
Partition type:
  p  primary (0 primary, 0 extended, 4 free)
  e  extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151):
Using default value 2097151

Command (m for help): 
```

- 파티션 타입 지정 : 기본 파티션은 primary로 설정하여 줍니다.
- 파티션 번호 설정 : 처음 생성하는 파티션이기 때문에 1로 설정해줍니다.
- 첫 섹터 : 기본값으로 설정해줍니다. 엔터를 누르면 진행됩니다.
- 마지막 섹터 : 역시 기본값으로 엔터를 눌러 진행합니다.

이제 파티션 만들기에 대한 정보 입력이 완료되었지만, 아직 실제로 디스크에 반영된 상태는 아닙니다.

3) w : 실제 디스크에 저장하기 (write)

w를 누르면 지금까지 입력한 정보들이 실제 디스크에 저장되게 됩니다.

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
zed@zed-VirtualBox:~$ 
```

3. 포맷을 통해 만든 파티션에 파일 시스템을 적용해봅시다.

리눅스에는 다양한 종류의 파일 시스템을 사용할 수 있는데, 현재로써는 ext4를 가장 많이 사용합니다. ext4 파일 시스템이 가진 특징에 대해서는 위키백과 Ext4(<http://ko.wikipedia.org/wiki/Ext4>)를 참조하면 자세히 알 수 있을 것입니다.

만들어진 파티션에 다음과 같이 입력함으로써 ext4 파일시스템으로 포맷을 할 수 있습니다.

```
:$ sudo mkfs.ext4 /dev/sdb1
```

입력하면 다음과 같이 포맷이 됨을 확인할 수 있습니다.

```
zed@zed-VirtualBox:~$ sudo mkfs.ext4 /dev/sdb1
[sudo] password for zed:
mke2fs 1.42.5 (29-Jul-2012)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
65536 inodes, 261888 blocks
13094 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
      32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

zed@zed-VirtualBox:~$
```

이제 우리가 sda 루트 드라이브를 사용하던 것처럼 이 드라이브도 자유롭게 사용할 수 있는 준비를 모두 마쳤습니다. 새롭게 드라이브를 실제로 추가할 경우도 위와 같은 방법으로 파티션을 나누고, 파일 시스템을 설정해준다면 쉽게 새 드라이브를 설정할 수 있을 것입니다.

그렇지만 아직 이 드라이브를 다른 디렉토리에 연결시켜주지 않았기 때문에, 이에 대한 마운트 작업이 필요합니다. 이에 대한 실습이 바로 이어집니다.

4. 특정 디렉토리에 디스크를 마운트해봅시다.

리눅스에서는 어떤 디렉토리에나 디스크를 마운트할 수 있습니다. 마운트를 하면, 이미 존재하던 디렉토리 속의 내용은 마운트를 해제하기 전까지 보이지 않습니다. 다음 명령어를 가지고 우리가 만든 파티션을 특정한 디렉토리에 마운트 시켜봅시다.

```
$ mkdir ~/myd # 홈 디렉토리에 myd라는 디렉토리를 생성합니다.  
$ sudo mount -t ext4 /dev/sdb1 myd # myd 디렉토리에 sdb1 파티션을 연결합니다.  
$ df # 현재 마운트 상태를 확인합니다.
```

```
zed@zed-VirtualBox:~$ mkdir ~/myd  
zed@zed-VirtualBox:~$ sudo mount -t ext4 /dev/sdb1 myd  
[sudo] password for zed:  
zed@zed-VirtualBox:~$ df  
Filesystem 1K-blocks Used Available Use% Mounted on  
/dev/sda1 15349744 3687632 10875732 26% /  
none 4 0 4 0% /sys/fs/cgroup  
udev 1024336 1024332 1% /dev  
tmpfs 206508 776 205732 1% /run  
none 5120 0 5120 0% /run/lock  
none 1032536 80 1032456 1% /run/shm  
none 102400 12 102388 1% /run/user  
/dev/sdb1 1014680 1284 961020 1% /home/zed/myd  
zed@zed-VirtualBox:~$
```

이와 같이 정상적으로 마운트 작업이 완료되었다고 하더라고, 마운트 후에 이 장치의 소유자가 루트이기 때문에 일반 유저로는 정상적으로 접근이 어렵습니다. 때문에 소유주를 바꾸는 작업이 필요한데, 이 때 쓰는 명령어가 바로 chown입니다. 다음과 같이 입력해봅시다.

```
$ sudo chown -R zed myd # zed 부분에 본인의 사용자 이름
```

그런 뒤에 [ls -l]을 입력하여 권한을 확인해봅시다.

```
-rw-rw-r-- 1 zed zed 940 11월 7 23:03 main.o  
drwxr-xr-x 3 zed root 4096 1월 5 21:20 myd  
-rw-rw-r-- 1 zed zed 64 11월 7 23:03 myprint.h
```

[ls -l]의 경우 순서대로 권한, 링크수, 유저, 그룹, 크기, 시간, 파일이름을 나타내줍니다. 실행해보면 위와 같이 소유주가 zed로 변경되었음을 알 수 있습니다.

드라이브의 사용을 모두 완료하고 나서 마운트를 해제할 일이 있을 수 있는데, 마운트를 해제하는 과정은 다음과 같이 명령어 하나만 입력해주면 됩니다.

```
$ sudo umount /dev/sdb1
```

다시 [df]를 통해 마운트 상태를 확인하면 마운트가 해제되었음을 알 수 있으며, 디렉토리의 내용 역시 다시 보이는 것을 확인할 수 있을 것입니다.

Advanced

권한(permission)에 대해 알아봅시다.

앞서 우리는 소유자를 변경하는 chwon 명령어에 대해 배웠습니다. 파일의 소유자만큼이나 보안에 중요한 것이 그 파일이나 디렉토리에 대한 읽기, 쓰기, 실행 권한을 조정하는 것입니다. [ls -l] 명령을 실행했을 때 나오는 권한정보를 통해 우리는 해당 파일이 어떤 권한을 가지고 있는지 알 수 있습니다.

1) 10자리 중 첫 자리는 파일의 특성을 나타내며, 다음과 같은 종류가 있습니다.

- d: 디렉토리
- l : 링크
- -: 일반 파일
- b: 블록 장치 (디스크)
- c: 캐릭터 장치 (마우스, 키보드)

2) 2번째 자리부터는 세 자리씩 끊어서 각각 파일을 소유한 사용자의 권한, 그룹의 권한, 그리고 기타 사용자의 권한을 나타내며, r, w, x로 구분합니다.

- r: 읽기 가능 (숫자 4에 대응)
- w: 쓰기 가능 (숫자 2에 대응)
- x: 실행 가능 (숫자 1에 대응)

3) 권한 정보에 대한 예제는 다음과 같습니다.

- rwxr-x-- (750) : 소유자는 모든 권한, 그룹은 읽기/ 실행권한
- r--r--r-- (400) : 모든 사용자에게 읽기 권한만 부여
- rwxrwxrwx (777) : 모든 사용자에게 모든 권한

4) 권한 정보를 바꾸기 위해서는 chmod 명령을 사용합니다.

```
$ touch a.txt
$ chmod 400 a.txt
$ echo "Hi, again" >> a.txt #error
```

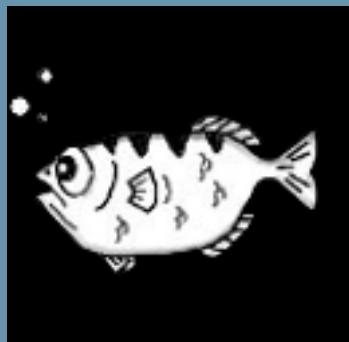
위는 권한 설정을 400으로 한 뒤에 a.txt를 수정하려 하면 오류가 발생하는 예제입니다.

Extra Section.

GDB

gdb는 콘솔에서 사용할 수 있는 강력한 디버거로, gcc와 함께 사용하기에 좋습니다. 이번 시간에는 gdb의 기초 사용법에 대해 이해하고, 프로그램의 오류 원인을 알아내는 것을 목표로 합니다.

Supplies



GDB

보통은 GDB라고 부르는 GNU 디버거는 GNU 소프트웨어 시스템을 위한 기본 디버거입니다. GDB는 다양한 유닉스 기반의 시스템에서 동작하는 이식성있는 디버거로, 에이다, C, C++, 포트란 등의 여러 프로그래밍 언어를 지원하고 있습니다.

Lecture

gdb를 사용하기 위한 기초 준비를 시작합니다.

1. 디버깅과 디버거가 무엇인지 알아봅시다.

디버그(debug), 디버깅 (debugging) 혹은 수정은 컴퓨터 프로그램의 정확성이나 논리적인 오류(버그)를 찾아내는 테스트 과정을 말합니다. 일반적으로 디버깅을 하는 방법으로 테스트 상의 체크, 기계를 사용하는 테스트, 실제 데이터를 사용해 테스트하는 법이 있습니다.

디버거(debugger)는 디버그를 돋는 도구입니다. 디버거는 주로 원하는 코드에 중단점을 지정하여 프로그램 실행을 정지하고, 메모리에 저장된 값을 살펴보며, 실행을 재개하거나, 코드를 단계적으로 실행하는 등의 동작을 하고 있습니다. 고급 디버거들은 메모리 충돌 감지, 메모리 누수 감지, 다중 스레드 관리 등의 기능도 지원합니다.

한 예로, 다음과 같은 오류들이 떴는데, 이 오류가 어디서 발생했는지를 우리 눈으로 파악하는 것은 매우 힘든 일입니다.

```

1 #include <stdio.h>
2
3 int *sum(int x, int y);
4
5 int main() {
6     int *p, *q;
7     p = sum(10,20);
8     q = sum(50,60);
9     printf("%d\n", *p); //30
10    printf("%d\n", *q); //110
11 }
12
13 int *sum(int x, int y) {
14     int ret = x + y;
15     return &ret;
16 }
```

```
hosu13@hosu13-VirtualBox ~/linux132/week4/gdb_ex $ ./a.out
110
-1216521928
```

디버거는 다음과 같은 오류들을 파악하는 데 큰 도움을 줍니다. 이러한 디버거를 사용하기 위해서는 우리가 실행하려는 파일에 디버깅을 하기 위한 정보가 포함되어 있어야 합니다. 앞서 gcc에 대해 배울 때 우리는 실행 파일에 디버깅 정보를 넣는 방법에 대해 배웠습니다.

디버깅 정보와 경고를 포함하기 위해서는 다음과 같은 명령어를 사용할 수 있습니다.

```
$gcc -g -Wall -o test file1.c file2.c file3.c
```

이에 대한 자세한 내용은 5장 GCC를 참고하기를 바랍니다.

2. gdb를 사용해봅시다.

gdb를 사용하는 방법은 크게 다음과 같은 3가지로 나누어집니다.

- 1) 일반적인 방법은 다음과 같습니다.

```
:$gdb <실행파일이름>
```

- 2) 실행중인 프로그램을 디버깅할 때

```
:$gdb <실행파일이름> <pid>
```

- 3) 덤프만 남기고 죽은 프로세스를 디버깅할 때

```
:$gdb <실행파일이름> <코어덤프이름>
```

다음은 일반적인 방법으로 gdb를 실행한 모습입니다.

```
터미널 - zed@zed-VirtualBox: ~
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
zed@zed-VirtualBox:~$ gdb test.c
GNU gdb (GDB) 7.5.91.20130417-cvs-ubuntu
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
"/home/zed/test.c": not in executable format: File format not recognized
(gdb) ■
```

gdb를 종료하기 위해서는 [q]나 [Ctrl + D]를 눌러주면 됩니다.

```
(gdb) q
zed@zed-VirtualBox:~$ ■
```

3. 소스코드를 보는 방법에 대해 알아봅시다.

gdb 프롬프트상에서 다음과 같이 입력하면 소스코드를 확인할 수 있습니다.

```
(gdb) l [줄번호] | [함수이름]
```

다음과 같이 사용할 수 있습니다. main함수가 존재하는 코드를 가지고 함께 따라해봅시다.

```
(gdb) l
```

```
(gdb) l
1 #include "diary.h"
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[]) {
5     //local variables
6     int n = 0;
7     int x = 10;
8
9     if(argc != 2)
10 }
```

[l]을 입력할 경우 다음과 같이 처음부터 10줄이 나오게 됩니다. 나머지 부분은 [Enter]키를 누르면 다시 10줄이 보이게 됩니다.

```
(gdb) l main
```

```
(gdb) l main
1 #include "diary.h"
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[]) {
5     //local variables
6     int n = 0;
7     int x = 10;
8
9     if(argc != 2)
10 }
```

[l main]을 입력할 경우 main함수와 관련된 부분이 나타나게 됩니다.

```
(gdb) l 22
```

[l 22]를 입력할 경우 22번째 라인과 관련이 있는 부분부터 밸춰되어 나오게 됩니다.

gdb에서 가장 중요한 명령어는 바로 [Enter]입니다. 이 [Enter]는 앞 명령을 진행시키기도 하지만, 명령이 끝났을 때 해당 명령을 반복해서 수행하기도 합니다. 한 예로, 위의 [l 22]가 모두 끝난 뒤 [Enter]를 누르면 다시 해당 명령어가 실행되게 됩니다.

4. 브레이크 포인트(중단점) 기능을 사용할 수 있습니다.

Eclipse나 Visual Studio와 같은 프로그램에서 제공하는 중단점 기능은 디버깅 과정에서 매우 유용합니다. GDB에서도 이러한 중단점 기능을 그대로 사용할 수 있습니다. 프로그램을 순차적으로 실행하다가 브레이크 포인트를 만나면 일시정지가 되도록 하는 기능입니다.

다음과 같은 명령들을 가지고 중단점을 설정할 수 있습니다.

```
(gdb) b <줄번호> | <함수이름> # 함수나 줄 단위로 중단점을 설정할 수 있습니다.  
(gdb) info b #설정된 중단점의 정보를 보여줍니다.
```

역시 main 함수가 있는 프로그램을 가지고 명령을 실행해봅니다.

```
(gdb) b main  
(gdb) b 19  
(gdb) info b
```

여기까지 입력하면, 두 개의 중단점을 설정하고 그 중단점을 다음과 같이 보여줍니다.

```
(gdb) b main
Breakpoint 1 at 0x8048eea: file main.c, line 6.
(gdb) b 19
Breakpoint 2 at 0x8048f44: file main.c, line 19.
(gdb) info b
Num      Type            Disp Enb Address      What
1      breakpoint        keep y 0x08048eea in main at main.c:6
2      breakpoint        keep y 0x08048f44 in main at main.c:19
(gdb)
```

이제 [run] 명령을 입력하여 디버거 상에서 프로그램을 실행할 수 있습니다.

```
(gdb) run <인자> #인자는 프로그램에서 요구할 경우 선택사항
```

다음과 같이 main함수 앞에서, 중단점이 걸립니다. 그렇지만 [Enter]를 누르게 되면 다시 run동작을 실행하게 됩니다. 중단점을 진행시키기 위해서는 이를 위한 다른 명령어들이 필요합니다.

```
Starting program: /home/zed/linux132/week4/gdb_ex/diary
Breakpoint 1, main (argc=1, argv=0xbfffff194) at main.c:6
6          int n = 0;
(gdb)
```

다음과 같은 명령어들을 가지고 중단점을 진행할 수 있습니다.

```
:gdb) s #한 줄을 지나갑니다. 함수를 만나면 들어갑니다.  
:(gdb) s 5 #다섯 줄을 지나갑니다.  
:(gdb) n #한 줄을 지나가지만, 함수를 만나면 통과합니다.  
:(gdb) c #다음 중단점까지 계속 진행합니다.  
:(gdb) u #loop가 있을 시 그것을 바로 빠져나갑니다.  
:(gdb) finish #진행중인 함수 밖으로 나옵니다.
```

이와 같이 GDB에서는 GUI 기반의 개발 환경에서 제공하는 중단점과 흡사한 기능들을 제공하며, 사용법을 알면 오히려 GUI 환경보다 편할 수도 있습니다. 다음과 같이 [run]을 실행한 뒤에 자유롭게 중단점을 진행시켜 봅시다.

```
(gdb) run babo
Starting program: /home/zed/linux132/week4/gdb_ex/diary babo

Breakpoint 1, main (argc=2, argv=0xbfffff194) at main.c:6
6           int n = 0;
(gdb) s 2
9           if(argc != 2)
(gdb) s 2
Hi, babo

Breakpoint 2, main (argc=2, argv=0xbfffff194) at main.c:19
19          while (n <= 10000)
(gdb) c
Continuing.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 5
7 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
```

중단점을 삭제할 때에는 [d] 명령어를 이용합니다.

```
:gdb) d <고유번호>
#고유번호는 [info b]를 통해 나오는 중단점의 번호를 말함
```

또한, 이러한 중단점이 실행되기 위한 조건을 걸 수도 있습니다. 이는 매우 유용한 기능인데, 오류가 나는 부분만 선별적으로 중단점을 실행시켜 번거로운 동작을 줄일 수 있기 때문입니다. 조건을 거는 방법은 중단점을 만들 때 우리가 알고 있는 조건식을 써 주면 됩니다.

```
:gdb) b main if n > 100
:gdb) b 19 if n < 20
```

이와 같이 작성하고 [run]을 할 시에 중단점은 if절 이하가 만족할 때에만 작동됩니다.

5. 기타 다양한 기능은 다음과 같습니다.

1) 와치 포인트

프로그램이 run되고 있을 때, 중단점에 위치하는 순간 특정 변수의 값을 프로그램 진행의 key로 설정할 수 있습니다. 이를 와치 포인트라고 합니다. 역시 조건식과 함께 설정하며, 조건식이 만족될 경우 프로그램이 멈추는 것으로, while문과 비슷한 유형이라고 할 수 있습니다. 이 와치 포인트는 다음과 같이 설정할 수 있습니다.

```
:gdb) watch n if n > 9000
```

이는 n이라는 변수가 9000이상이 될 경우 프로그램을 중단하라는 의미입니다.

```
터미널 - zed@zed-VirtualBox: ~/linux132/week4/gdb_ex
파일(F) 편집(E) 보기(V) 터미널(T) Tabs 도움말(H)
8765 8766 8767 8768 8769 8770 8771 8772 8773 8774 8775 8776 8777 8778 8779 8780
8781 8782 8783 8784 8785 8786 8787 8788 8789 8790 8791 8792 8793 8794 8795 8796
8797 8798 8799 8800
8801 8802 8803 8804 8805 8806 8807 8808 8809 8810 8811 8812 8813 8814 8815 8816
8817 8818 8819 8820 8821 8822 8823 8824 8825 8826 8827 8828 8829 8830 8831 8832
8833 8834 8835 8836 8837 8838 8839 8840 8841 8842 8843 8844 8845 8846 8847 8848
8849 8850 8851 8852 8853 8854 8855 8856 8857 8858 8859 8860 8861 8862 8863 8864
8865 8866 8867 8868 8869 8870 8871 8872 8873 8874 8875 8876 8877 8878 8879 8880
8881 8882 8883 8884 8885 8886 8887 8888 8889 8890 8891 8892 8893 8894 8895 8896
8897 8898 8899 8900
Hardware watchpoint 3: n

Old value = 9000
New value = 9001
main (argc=2, argv=0xbfffff194) at main.c:19
19          while (n <= 10000)
(gdb) █
```

2) 변수값 출력

[p]와[display] 명령어를 통해서 변수의 값을 볼 수 있습니다.

```
:gdb) p <변수명>
# 구조체, * 연산자, 형변환 등 C언어에서 사용하는 방법을 그대로 사용할 수 있음

:gdb) display <변수명>
# 매번 자동으로 변수값을 출력, 해제는 [undisplay]
```

```
(gdb) display n
1: n = 9001
(gdb) c
Continuing.
Hardware watchpoint 3: n

Old value = 9001
New value = 9002
main (argc=2, argv=0xbfffff194) at main.c:19
19          while (n <= 10000)
1: n = 9002
(gdb) c
Continuing.
Hardware watchpoint 3: n
```

6. 코어 덤프는 프로그램이 비정상적으로 종료될 때 남기는 파일입니다.

코어 덤프는 프로그램이 죽기전에 남기는 파일로, 다음과 같은 상황에서 생길 수 있습니다.

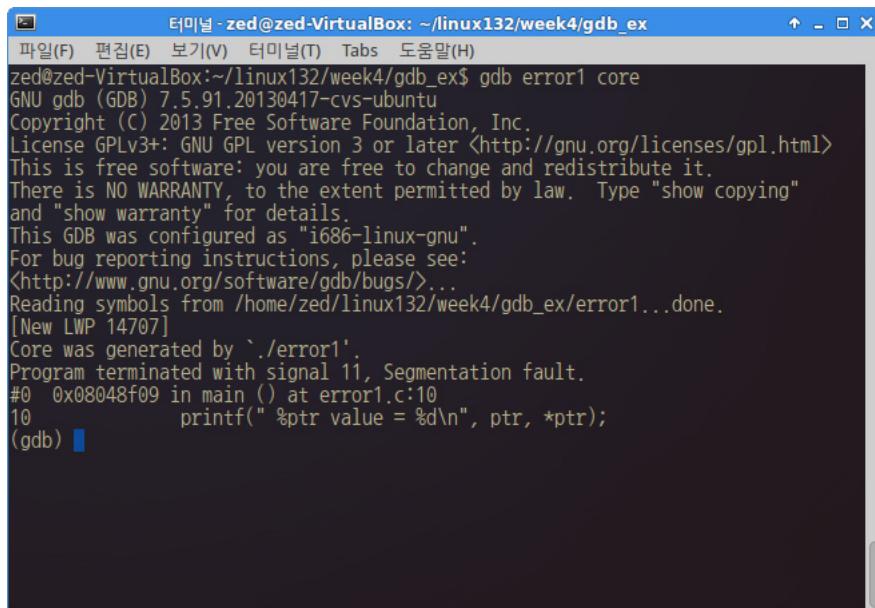
- 1) 키보드로 종료 명령을 내렸을 때
- 2) 잘못된 명령을 수행했을 때
- 3) 기타 오류나 인터럽트로 인해 종료되었을 때

이번에는 실험을 위해 오류가 나는 파일을 실행해보도록 하겠습니다.

```
zed@zed-VirtualBox:~/linux132/week4/gdb_ex$ ./error1
0x9469718 value = 50
세그멘테이션 오류 (core dumped)
zed@zed-VirtualBox:~/linux132/week4/gdb_ex$ ls
Makefile  calendar.o  diary  error1  loop  main.c  memo.c
calendar.c core        diary.h  error1.c  loop.c  main.o  memo.o
```

다음과 같이 [core]파일이 형성되었음을 알 수 있습니다. 가장 처음에 말하였듯이, 코어 덤프를 이용하여 error1을 디버깅하기 위해서는 다음과 같이 gdb를 실행해주어야 합니다.

```
:$gdb <실행파일이름> <코어덤프이름>
:$gdb error1 core
```



위와 같이 어느 부분에서 오류가 나게 되는지 디버거가 알려주게 됩니다. 이를 토대로 우리가 지금까지 배운 것을 기반으로 하여 디버깅 작업을 할 수 있습니다.

GDB는 생각보다 어렵지 않고, 매우 유용합니다. 이는 다른 디버거도 마찬가지입니다. 이와 같은 디버거를 능숙하게 사용하는 것은 오류를 잡는데 소요되는 시간을 크게 줄여줍니다.

[교재를 봐주시는 분들께 감사드립니다.]

Extra Section. End