젠투 데스크탑(64bit) 설치 도움 문서

가장 좋은 것은 공식 문서입니다. 분 문서는 참고로 활용하고 공식 문서를 늘 우선으로 참고하시기를 바랍니다.

시작하기

문서를 읽는 분들은 어느 정도 리눅스를 사용한 경험이 있으신 분들을 대상으로 합니다. 만일 자신이 전혀 리눅스를 알지 못하고, 어제까지는 MS 윈도우를 이용하다 오늘은 리눅스를 사용하리라 마음먹으신 분들께는 젠투 리눅스를 **추천드리지 않습니다**. 보통 그러한 분들은 우분투 리눅스를 추천해 드립니다. 한편 어느 정도 리눅스에 익숙해졌고, 보다 이런저런 재미(?)를 느끼고 싶은 분들께는 아치 리눅스를 권해 드립니다. 젠투 리눅스는 좀더 리눅스로 도전을 삼질을 즐기시고 싶으신 분들께 권합니다. 한편 리눅스를 공부하시려는 분들께는 적극 추천합니다. 시스템이 운영되는 모습을 보다 가까이에서 지켜볼 수 있습니다.

솔직히 말해 저 또한 '바퀴를 재발명'하는 귀찮은 일을 하기가 꺼려집니다. 다시 말해 자세한 젠투 설치 문서를 작성하는 것은 별 의미가 없다고 판단합니다. 또한 자잘한 설정 정도는 공식 문서를 참고하는 것이 좋습니다. 이 문서는 각 단계에서 참고가 될 만한 사항을 부가적으로 정리해 두는 문서에 불과합니다.

아키텍쳐 고르기

저는 젠투를 일반적인 윈도우에서 사용하던 데스크탑 PC를 리눅스으로 활용한다고 가정하고 본 문서를 진행하도록 하겠습니다. 그러므로 x86이나 amd64 (x86_64) 중 하나가 되겠죠. 몇 년 전이라면 32비트인 x86을 이용했을지도 모르지만, 이제는 64bit인 amd64를 사용해야죠. 그러므로 문서는 amd64를 기준으로 서술됩니다.

설치 CD 구하기

설치 CD는 홈페이지 메뉴의 'Get Gentoo!' 메뉴를 클릭하면 나오는 페이지에서 쉽게 구할 수 있습니다. amd64의 'iso' 링크를 선택해 다운로드 받을 수 있습니다. 일반적인 배포판의 설치 CD(DVD)는 적게는 600메가바이트에서 많게는 2~3기가바이트까지 차지하는 것에 비해 200메가바이트를 넘지 않는 가뿐한 용량입니다. 젠투는 다음, 카이스트 등에서 미러링 서비스를 하고 있습니다. 특히 다음 미러 서버는 이용하기 매우 쾌적합니다.

다음 미러 서버에서 ISO 구하기

웹브라우저로 http://ftp.daum.net/gentoo/releases/amd64/current-iso/ 에 접속합니다.
install-amd64-minimal-YYYYMMDD.iso 파일을 다운로드 받으면 됩니다.

USB 부팅 디스크 만들기

최근에는 CD-ROM을 이용하는 일이 많이 줄었습니다. 대개 ISO의 파일을 USB로 옮겨 부팅을 하는 일이 더 많아졌지요. 역시 가장 안정적인 방법은 CD로 구워서 CD로 부팅하는 방법입니다만, USB를 활용하시고 싶다면 unetbootin, LinuxLive USB Creator를 활용해여 리눅스 부팅 가능한 USB로 만드시면 됩니다. 가끔씩 이렇게 만든 USB로는 올바르게 부팅이 안 되는 경우도 있는데, 이럴 때는 USB에 CD영역을 만드는 방법을 추천드립니다.

젠투 설치 CD로 부팅

이제 PC를 끄고 젠투로 부팅을 시작합니다. 아마 왠만한 트러블이 없다면 잘 부팅이 될 것입니다. 왠만한 배포판이 화려한 설치 관리자를 제공하는 반면, 여전히 젠투는 시커먼 콘솔 화면이 여러분을 반갑게(?) 맞이할 것입니다. 그래도 너무 겁먹지 마세요 그리고 앞으로 충분히 친숙해질 것입니다.

가끔 부팅 단계에서부터 성공적이지 못해 애를 먹는 경우가 있습니다. 일단 부팅부터 잘 안되다고 해서 지레 포기하진 말고, 적절한 증상을 구글에 검색해 해결을 모색하는 것이 좋습니다. 부팅조차 잘 되지 않는 이유는 아래 몇 가지 중 하나일 수 있습니다.

1. ISO 파일 자체부터 문제가 있다.
   심심찮게 일어나는 일이기도 합니다. 리눅스 배포판 ISO를 받아 설치하기 전에는 반드시 ISO 파일이 올바른지부터 먼저 확인하는 습관을 들이세요. 아까 ISO 파일을 받기 전 이름은 동일한데 이름 끝에 '.CONTENTS', '.DIGETST', '.DIGESTS.asc'라는 문자가 추가적으로 각각 붙은 세 개의 파일을 보셨을 것입니다. 이 세 파일은 ISO 파일이 올바른지 아닌지를 증명해주기 위해 보조적으로 존재합니다. ISO가 올바른지 아닌지 확인하는 방법은 공식 문서 부분을 참고하세요. 윈도우에서는 간단하게 FCIV 같은 툴로 검사를 하면 됩니다.
2. USB 혹은 CD의 제작에서 결함이 생겼다
   이 또한 가능한 원인 중 하나입니다. USB 같은 경우 잘못 만들어지는 경우가 상당히 많습니다. USB 부팅 프로그램은 대개 ISO 파일의 내용을 플래시 드라이브로 적절히 복사를 합니다. 그러므로 ISO의 구조는 플래시 드라이브에 생성된 구조와 100% 일치하지 않습니다. 이렇게 약간 변화된 부분이 적절하지 않다면 문제가 발생하겠죠. 가장 안정적인 방법은 아예 CD로 만드는 것이고, 그렇게 하기 어렵다면 USB에 CD영역을 그대로 만드는 것이 좋습니다. 만일 CD나 CD 영역을 생성해서 만든 플래시 드라이브가 여러 PC에서도 똑같이 이상 작동을 보인다면 분명히 CD나 플래시 드라이브에 문제가 있는 것이겠죠.
3. 사용하는 PC의 부품이 리눅스 커널이 잘 지원하지 않는다.
   이런 일은 이제 거의 드뭅니다. 그렇지만 없다고 볼 수도 없지요. 사실 리눅스 커널이 잘 지원하지 않는다면 어떤 리눅스로 부팅을 하더라도 문제가 발생하겠지만 또 어떤 PC에서는 멀쩡히 잘 부팅되는 경우도 있습니다. 이 경우는 어떤 증상인지에 따라 적절히 대응을 해야 합니다. 보통은 부팅할 때 커널 옵션을 변경함으로써 해결하는 것이 대부분입니다. 커널 옵션은 이 곳에 잘 나와 있습니다. 이런 귀찮은 상황이 일어나지 않기를 바랄 분입니다.

부팅 중에는 키보드를 선택해야 한다는지 하는 몇몇 입력이 있습니다. 대부분의 경우는 그냥 기본값인 'US' 키보드를 사용해도 무방하므로 적절히 엔터를 치면 됩니다. 부팅이 올바르게 되면 붉은 색 샵(#) 기호가 보이고, 커서가 깜빡일 것입니다. 일단 축하합니다. 무사히 젠투로 부팅을 할 수 있다는 것을 보여 주었습니다.

루트 계정에 대해

지금은 여러분들이 루트(root) 계정을 이용하고 있습니다. 루트는 한 리눅스 시스템에서 가장 높은 권한을 가진 계정입니다. 비유하자면 한 집안의 큰어른입니다. 루트의 명령은 거의 절대적입니다. 그렇기에 시스템을 원하는대로 주무를 수 있는 편리하고 강력한 계정이지만, 실수하면 시스템을 홀랑 말아먹을 수도 있는 위험한

계정입니다. 일반적인 상황에서는 루트 계정으로 작업을 계속하는 것을 절대 권하지는 않지만, 지금 우리는 시스템을 설치해야 하는 과정에 있으므로 계속 루트를 사용하는 것이 더 적절합니다. 보통 리눅스 명령어를 사용하는 예에서 루트 계정이 필요한 명령어는 앞에 '#' 기호를 붙입니다: 일반 계정의 경우는 '$'를 붙이지요. 이 문서에서도 이러한 표기를 사용할 것입니다. 바로 아래처럼 말이죠.

```
# command
```

명령어 표기에 관해

위의 예시와 같이 박스 내부의 문장은 쉘에 입력할 명령어입니다. 여러분은 # 후의 문자를 직접 쉘에 타이핑해 적어야 합니다. 그러나, '<'와 '>' 안에 있는 문자는 여러분의 시스템의 상황에 맞게 적절히 변화시켜 적기 바랍니다. 일례로,

```
# cp <path_from>/sample <path_to>/sample_copied
```

라는 예는 'path_from'의 sample이란 파일을 'path_to'에 'sample_copied'라는 파일로 복사하라는 명령인데, 'path_from'과 'path_to'는 여러분들이 시스템에 맞게 고쳐 적어야 합니다. 대개 여러분들이 이전 단계에서 스스로 결정한 사항입니다.

가끔 한 명령어의 우측에 샵 기호가 다시 등장할 때가 있습니다. 그것은 그 명령어가 가진 의미를 전달하기 위한 주석입니다. 그 주석까지는 타이핑할 필요가 없습니다. 가령,

```
# cat 'Asia/Seoul' > /etc/timezone # 시간대를 설정
```

에서 '# 시간대를 설정' 이란 부분은 이 명령이 하는 의미를 상술한 것일 뿐이므로 굳이 적을 필요는 없습니다. 또한 명령어 이후에 '#' 기호 없이 텍스트를 적을 수 있는데, 이것은 둘 중 하나입니다. 시스템이 유저에게 출력하는 메시지이거나, 유저가 시스템에게 입력해야하는 메시지입니다. 그때그때 상황에 맞춰 구분할 수 있으므로 혼동의 여지는 없다고 봅니다.

자신의 **PC** 사양을 확인하기

윈도우를 사용하시는 분들, 혹은 리눅스를 사용하시는 분들도 자신의 하드웨어가 정확히 어떻게 되는지는 잘 알지 못할 것입니다. 아니, 이런 걸 일일이 다 알아야 한다면 PC를 사용하는 것 자체가 고역이 되어 버릴 것입니다. 그렇지만 젠투를 사용하려면 어느 정도는 확인해 두는 것이 좋습니다. 왜냐하면 여러분들은 스스로 '커널'도 생성해 내야 하니까요!

CPU, 램, HDD, 사운드카드, 모니터, 키보드, 마우스, 프린터 등등의 여러 장비의 목록을 확보해 두는 것이 좋습니다. 요즘이야 왠만하면 리눅스가 인식 못하는 일이 드물긴 합니다만, 정보를 확실히 모아두는 작업을 먼저 해 두는 것이 앞으로의 설치 작업은 물론, 앞으로의 시스템 관리에도 도움이 됩니다.

저 같은 경우 하드웨어에 대한 정보는 lspci 정도면 충분하다고 생각합니다. 일단 다음 명령을 수행합니다.

```
# cd ~
# lspci -v > lspci.txt
```

이 명령어를 통해 PCI 장치의 정보를 각각 텍스트 파일로 볼 수 있습니다. lspci.txt에 나오는 정보는 꽤나

중요합니다. PC 장치 전반에 대한 정보가 출력되어 나오거든요. 게다가 -v 스위치를 주었기 때문에 젠투 설치 CD 가 가진 커널이 어떤 모듈을 사용하고 있는지에 대해 정보를 파악할 수 있으므로 나중에 커널을 컴파일할 때 매우 중요한 사전정보가 됩니다. 참고로 저의 lspci -v 정보의 일부를 나열하면 다음과 같습니다.

```
00:00.0 Host bridge: Intel Corporation Core Processor DMI (rev 11)
        Subsystem: ASRock Incorporation Device d131
        Flags: fast devsel
        Capabilities: [40] #00 [0000]

00:03.0 PCI bridge: Intel Corporation Core Processor PCI Express Root Port 1
(rev 11) (prog-if 00 [Normal decode])
        Flags: bus master, fast devsel, latency 0
        Bus: primary=00, secondary=07, subordinate=07, sec-latency=0
        I/O behind bridge: 0000e000-0000efff
        Memory behind bridge: f8000000-fbffffff
        Prefetchable memory behind bridge: 00000000d4000000-00000000dfffffff
        Capabilities: [40] Subsystem: ASRock Incorporation Device d138
        Capabilities: [60] MSI: Enable- Count=1/2 Maskable+ 64bit-
        Capabilities: [90] Express Root Port (Slot+), MSI 00
        Capabilities: [e0] Power Management version 3
        Capabilities: [100] Advanced Error Reporting
        Capabilities: [150] Access Control Services
        Capabilities: [160] Vendor Specific Information: ID=0002 Rev=0
Len=00c <?>
        Kernel driver in use: pcieport
        .....
```

출력 메시지는 매우 길고 번잡한 정보가 많으니 일일이 주의깊게 살펴 보아야 할 필요는 없습니다. 아주 간결하게 커널 드라이버 이름만 사전순으로 출력하려면 다음 명령을 응용하는 것도 좋습니다.

```
# lspci -v | grep "use:" | sed 's/\s.*: //' | sort -u
ahci
ata_piix
ehci-pci
firewire_ohci
i7core_edac
i801_smbus
lpc_ich
nvidia
pata_jmicron
pcieport
r8169
snd_hda_intel
```

이 목록은 메모해서 기록해 두는 것이 좋습니다.

네트워크 설정

젠투 리눅스의 설치용 ISO가 그토록 적은 용량으로도 배포가 가능한 것은 설치에 꼭 필요한 요소만 남겨두기 때문입니다. 시스템에 직접 설치를 할 것들은 모두 인터넷을 통해 받아와 시스템에서 직접 생성해냅니다. 설치 CD는 설치에만 필요한 여러 공구/장비만 담고 있으며, 설치에 필요한 이외의 것은 인터넷으로 직접 공수해 옵니다. 그러므로 젠투 설치에 있어 네트워크는 상당히 중요합니다. 인터넷을 통해 프로그램을 받을 수 없다면 시스템의 설치는 요원한 일이겠지요. 그러므로 자신의 네트워크가 올바로 동작하는지 확실히 점검하고 넘어가야 합니다.

참고로 젠투는 반드시 설치 CD가 있어야만 되는 것은 아닙니다. 시스템에 기존의 리눅스가 설치되어 있다면 기존의 리눅스가 설치 씨디의 역할을 대신할 수도 있습니다. 물론 설치가 끝날 때 까지 기존의 시스템은 지워져선 안 되지만요. 상당히 독특하지요? 🙂

유선의 경우 (권장)

우선 설치는 안정적으로 통신을 할 수 있는 유선 장비를 통해 하는 것을 권합니다. 무선의 경우 장비의 종류에 따라 리눅스에서 잘 지원이 되지 않는 경우도 있고, 보안을 위해 비밀번호 등을 설정하는 과정도 거쳐야 합니다.

공식 문서에는 여러 환경을 고려하여 상세한 설명을 하고 있지만, 이 문서에서는 보통 일반적인 경우를 가정해서 문서를 작성하도록 하겠습니다. 말하자면 사용자는 보통의 인터넷 회선을 사용중이고, 추가적으로 유/무선 공유기를 활용하고 있다고 가정하겠습니다. 그러므로 설치를 하는 PC가 직접적으로 인터넷 회선에 연결되지는 않습니다. 또한 아직은 IPv4를 사용한다고 가정하겠습니다.

보통의 경우 굳이 IP를 설정하지 않고 공유기에서 자동으로 IP를 받아오는 방법을 택할 것입니다. 이 경우 다음과 같은 명령어를 통해 IP를 확인해 볼 수 있습니다.

```
# ifconfig
eth0 Link .....
inet addr: 192.168.0.2 .....
.....
```

이렇게 'eth0'와 같은 문자가 보이면 네트워크 장비가 올바로 인식되었다는 것을 의미합니다. IP 주소는 192.168.0.2로 할당된 것을 확인할 수 있습니다. 만일 랜카드 장비가 1개 이상이면 그 수만큼 보이겠지요. 단, 주의할 점이 있습니다. 네트워크 장비 'lo'는 PC에 연결된 실제 네트워크 장비가 아닙니다.

만일 연결은 되었는데 IP를 받아오지 못했다면 다음 명령어로 주소를 할당받을 수 있습니다.

```
# dhcpd <device> # device는 'eth0' 같은 장치 이름입니다.
```

고정주소를 지정하기

만일 고정 주소를 선호한다면 다음과 같이 작업하면 됩니다. 일례로 eth0가 192.168.0.100의 주소로 할당하는 것을 원한다면 다음고 같이 작업하면 됩니다.

```
# ifconfig eth0 192.168.0.100 broadcast 192.168.0.255 netmask 255.255.255.0
# route add default gw 192.168.0.1
# nano -w /etc/resolv.conf
nameserver 168.126.63.1
nameserver 168.126.63.2
```

(작성후 Ctrl+x, 그리고 y를 누르고 엔터를 치면 저장됩니다)

- 192.168.**0**.100이라면 보통은 게이트웨이나 브로드캐스트 주소는 192.168.0.1, 192.168.0.255입니다.
- 넷마스크는 보통 255.255.255.0입니다.
- 168.126.63.1, 168.126.63.2는 KT의 DNS 주소입니다. resolv.conf에 적으면 됩니다.
  - 8.8.8.8, 8.8.4.4라는 구글의 DNS 주소도 있으니 KT를 원하지 않으시면 이 두 주소로 변경해도 무방합니다.
- 위 사항은 공유기의 설정에 따라 달라질 수 있으니 반드시 공유기 설정을 확인하고 값을 고쳐 주세요.

무선의 경우

무선의 경우 장비가 잘 인식되지 않는 일이 있습니다. 최신 하드웨어일수록 그러한 경우가 잦으며, 이 경우는 설치 단계에서 해결하기가 조금 곤란할 수도 있습니다. 이런 경우라면 그냥 유선을 사용하도록 하세요.

별로 권하는 방식은 아니지만 무선을 사용해야만 하는 경우도 있을 것입니다. 대개는 보안을 위해 암호를 걸어둘 것이고, 최소한 WEP를 사용하지는 않을 것입니다. (만일 WEP를 사용 중이라면 즉시 변경하기를 권고드립니다. WEP는 비밀 번호 알아내기가 정말 쉽습니다.) 개인적으로 이 비밀 번호 설정 또한 많이 귀찮습니다. 일단 설치 때만이라도 잠시 비밀번호를 해제하는 것도 좋은 방법입니다.

WPA를 이용하려면 wpa_supplicant가 필요합니다.

```
# nano -w /etc/wpa_supplicant/wpa_supplicant.conf
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1
network = {
  ssid="YOUR_SSID"
  scan_ssid=1
  psk="YOUR_PASSWORD"
  priority=1
}
```
(저장합니다. Ctrl+x를 누르고 y를 입력합니다.)

YOUR_SSID, YOUR_PASSWORD는 공유기에 맞는 무선 AP이름과 비밀번호를 넣어 주시면 됩니다. 다음 명령으로 무선 랜카드를 동작시킵니다. 무선 랜카드는 wlan0라고 가정합니다.

```
# wpa_supplicant -Bw -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
# dhcpd wlan0
```

고정주소를 원한다면 유선 네트워크에서 고정주소 지정하기를 참고하세요.

원격 접속 설정

젠투는 모든 패키지를 CPU가 알아서 컴파일해야 하므로 다른 배포판에 비해 상당히 많은 시간이 듭니다. 아마 컴퓨터를 켜 두고 외출을 하거나 아예 원격지에서 설치를 해야 하는 상황도 빈번히 있을 것입니다. 고정 IP를 강조한 것은 다름아닌 이러한 이유 때문인데, 원격지에서도 SSH를 통해 젠투의 설치를 계속할 수 있습니다. 설치 단계의 젠투에 원격으로 접속하려면 다음처럼 명령을 내려 SSH 서버를 활성화시키면 됩니다.

```
# passwd
```

```
New password: <패스워드>
Re-enter password: <패스워드 확인>
# /etc/init.d/sshd start
```

이렇게 해 두면 SSH 클라이언트를 통해 설치 중인 젠투에 언제든지 접속할 수 있습니다.

네트워크 확인

모든 설정이 잘 되어 있는지 확인해 봅니다. 네트워크는 아주 중요한 요소이므로 반드시 올바르게 동작해야 합니다.

```
# ping 192.168.0.1 # 게이트웨이, 즉 공유기와 통신해봅니다.
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=0.501 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=0.494 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=64 time=0.468 ms
...
Ctrl+c
```

이렇게 팔딱팔딱 메시지가 잘 들어온다면 공유기와의 연결, 그러니까 내부 연결망의 통신 상태는 양호한 것입니다. 이제 외부와의 연결도 양호한지 확인해 보도록 합니다.

```
# ping www.gentgoo.org # 젠투 서버는 약간 지연이 있을 수 있습니다.
PING www-bytemark-v4v6.gentoo.org (89.16.167.134) 56(84) bytes of data.
64 bytes from www.gentoo.org (89.16.167.134): icmp_seq=1 ttl=44 time=288 ms
64 bytes from www.gentoo.org (89.16.167.134): icmp_seq=2 ttl=44 time=288 ms
64 bytes from www.gentoo.org (89.16.167.134): icmp_seq=3 ttl=44 time=288 ms
...
# ping www.google.com #
PING www.google.com (173.194.72.99) 56(84) bytes of data.
64 bytes from tf-in-f99.1e100.net (173.194.72.99): icmp_seq=1 ttl=46
time=90.9 ms
64 bytes from tf-in-f99.1e100.net (173.194.72.99): icmp_seq=2 ttl=46
time=90.9 ms
64 bytes from tf-in-f99.1e100.net (173.194.72.99): icmp_seq=3 ttl=46
time=66.2 ms
```

만일 이 부분에서 제대로 응답이 오지 않고 타임아웃이 난다면, DNS 설정을 의심해 볼 필요가 있습니다. /etc/resolv.conf 파일을 열고 nameserver의 주소가 올바르게 기재되어 있는지 잘 확인해보세요. 위의 예처럼 내부망과 외부망의 연결이 잘 되었다면, 네트워크 설정은 순조롭게 끝마친 것입니다. 여러분의 젠투 PC는 이제 고립된 섬이 아닙니다!

디스크 준비하기

이제 젠투가 설치될 시스템의 디스크를 설정하는 단계입니다. 여기서는 디스크 파티션을 나누고, 파티션을 포맷한 후 마운트하는 작업을 수행합니다. 디스크를 수정하는 일은 언제나 주의에 주의를 거듭해 기울여야 합니다.

주의깊게 메시지를 보고 작업을 신중하게 진행하세요. 다른 장이라면 몰라도, 이 장의 거의 모든 명령어는 절대로 습관적으로나 반사적으로 키보드를 두드려서는 안 됩니다.

디스크는 설명의 편의를 위해 일반적인 하드디스크 하나를 적절히 나누어 사용한다고 가정합니다. LVM이나 SSD에 관한 설명은 생략하겠습니다. 또한 각 파티션의 크기는 2TB를 넘지 않는다고 생각하겠습니다. 이를 넘으면 GPT를 사용해야 하는데, 이에 관해서는 공식 문서를 참고하기를 바랍니다.

파티션 설정

시스템의 디스크와 파티션의 정보를 출력해봅니다.

```
# fdisk -l
Disk /dev/sda: 320.1 GB, 320072933376 bytes, 625142448 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 label type: dos
Disk identifier: 0x0008493a

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      206847      102400    7  HPFS/NTFS/exFAT
/dev/sda2          206848   446410751   223101952    7  HPFS/NTFS/exFAT
/dev/sda3       446412798   625141759    89364481    5  Extended
/dev/sda5       446414848   625141759    89363456   83  Linux
...
```

위는 어떤 시스템의 디스크 하나를 예로 든 것입니다. 윈도우는 장착된 디스크를 C, D, ... 로 표시하지만 리눅스에서는 sda, sdb, ... 와 같이 표시합니다. 예전에는 hda, hdb, ... 식으로 표기하기도 했습니다. 위 예의 경우 하드디스크 /dev/sda는 4개의 파티션으로 구성되어 있습니다.

첫번째 파티션은 sda1, 두번째 파티션은 sda2, 그리고 확장 파티션으로 sda3이 있으며 그 안에 sda5가 나머지 모든 공간을 차지합니다. sda1, sda2는 모두 주 파티션 (primary partition)입니다. 주 파티션은 디스크 당 총 4개밖에 설치할 수밖에 없습니다. 그러한 약점을 극복하고자 나온 것이 확장 파티션(extended partition)입니다.

sda1, sda2에는 현재 MS 윈도우가 설치되어 있습니다. MS 윈도우에 한해서 Boot flag이 붙습니다. sda1의 두 번째 열에 '*' 가 붙은 것은 확인할 수 있는데, 이것은 sda1이 바로 윈도우를 위한 부트 파티션이기 때문입니다. 보통 비스타 이후의 OS가 이런 스타일로 설치됩니다.

위의 예처럼 젠투가 설치될 시스템도 파티션을 나누어 관리하는 것이 좋습니다. 파티션의 개수는 유저가 정하기 나름입니다. 하나의 파티션에 모든 데이터를 관리하는 것도 좋지 않지만, 지나치게 많아도 관리하기 편하지 않습니다. 일반적인 데스크탑의 경우라면 아래와 같이 4~5개 정도의 파티션으로 나누는 것이 적당합니다. 총 1TB의 하드를 가지고 다음과 같이 파티션을 나눈다고 가정하겠습니다.

파티션	타입	크기	설명
/dev/sda1	ext4	32MB	부트 파티션입니다. 부팅에 필요한 파일만 관리하므로 크기가 클 필요가 없습니다.
/dev/sda2	swap	2GB	스왑 파티션입니다.
/dev/sda3			확장 파티션입니다.
/dev/sda5	ext4	32GB	루트 시스템 및 홈 디렉토리를 제외한 나머지를 위한 공간입니다. 적절히 조절하세요.

/dev/sda6	ext4	나머지 모두	사용자에게 가장 중요한 정보가 담기는 곳이므로 가장 넓은 공간을 둡니다.

부트 파티션(/dev/sda1)은 정말 부팅에 필요한 파일만 모아둡니다. 커널 컴파일에 집중하지 않는 한 이 파티션은 굳이 커야 할 이유가 없습니다. 루트 파티션(/dev/sda5)에는 여러 라이브러리와 프로그램들이 설치됩니다. 32GB 정도면 적당하지만, 이것저것 많이 설치해서 사용하고 싶다면 조금 더 안배하세요. 스왑은 '램의 2배를 할당해야 한다'는 룰이 있지만 꼭 지켜야 할 사항은 아닙니다. 예에서는 적당히 2GB를 잡았지만 더 늘여도 관계 없습니다. 그러나 지나치게 늘일 필요는 없습니다. 마지막으로 sda6는 홈 파티션만을 저장할 곳입니다. 이 파티션은 일반 사용자 소유의 파일들이 저장되는 곳이므로 이 곳의 용량은 가급적 가장 넉넉히 확보하는 것이 좋습니다.

파티션 만들기

실제로 파티션을 생성합니다.

```
# fdisk /dev/sda
```

sda는 사용하지 않은 새 디스크라고 가정합니다. 그러나 이 디스크에 기존의 파티션이 있다면 기존의 배치를 삭제하도록 합니다.

```
Command (m for help): p
Disk /dev/sda: 320.1 GB, 320072933376 bytes, 625142448 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 label type: dos
Disk identifier: 0x0008493a

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      206847      102400    7  HPFS/NTFS/exFAT
/dev/sda2          206848   446410751   223101952    7  HPFS/NTFS/exFAT
/dev/sda3       446412798   625141759    89364481    5  Extended
/dev/sda5       446414848   625141759    89363456   83  Linux

Command (m for help): d
Partition number (1-4): 1
...
```

이런 식으로 하나하나 파티션을 삭제할 수 있습니다. 이제 하나씩 파티션을 만들어보도록 합니다.

```
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-3876, default 1): (Hit Enter)
Using default value 1
```

```
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +32M
```

스왑은 이렇게 생성하면 됩니다.

1. Command로 'n'을 눌러 새 파티션 생성 명령 입력
   1. p를 눌러 주 파티션 설정
   2. 2 혹은 그냥 엔터를 쳐서 2번째 파티션으로 설정
   3. 처음 섹터는 그냥 엔터를 쳐서 기본값으로 설정
   4. '+2G'를 입력해 2GB의 크기 할당
2. Command로 't'를 눌러 파티션의 타입 조정
   1. Partition number는 2
   2. Hex code는 82

이렇게 루트 파티션, 확장 파티션, 그리고 홈 파티션을 전부 나누어 주면 됩니다. 그 다음 Command로 'w'를 눌러줍니다. 변화된 사항이 모두 디스크에 기록됩니다.

파일시스템 생성

각 파티션에 알맞게 파일시스템을 생성해줍니다. 전부 ext4를 사용합니다.

```
# mkfs.ext4 /dev/sda1
# mkfs.ext4 /dev/sda5
# mkfs.ext4 /dev/sda6
# mkswap /dev/sda2     # swap 생성
# swapon /dev/sda2     # swap 활성화
```

디스크 마운트

이제 디스크를 마운트해서 쉘을 통해 파일 및 디렉토리에 직접 접근할 수 있도록 해야 합니다.

```
# mount -v /dev/sda5 /mnt/gentoo
# mkdir -v /mnt/gentoo/{boot,home}
# mount -v /dev/sda1 /mnt/gentoo/boot
# mount -v /dev/sda6 /mnt/gentoo/home
```

파일이 전혀 없으므로 차이가 없을지 모르지만, 여기까지 에러 메시지가 없었다면 틀림없이 각 파티션이 각 디렉토리에 마운트 되었을 것입니다.

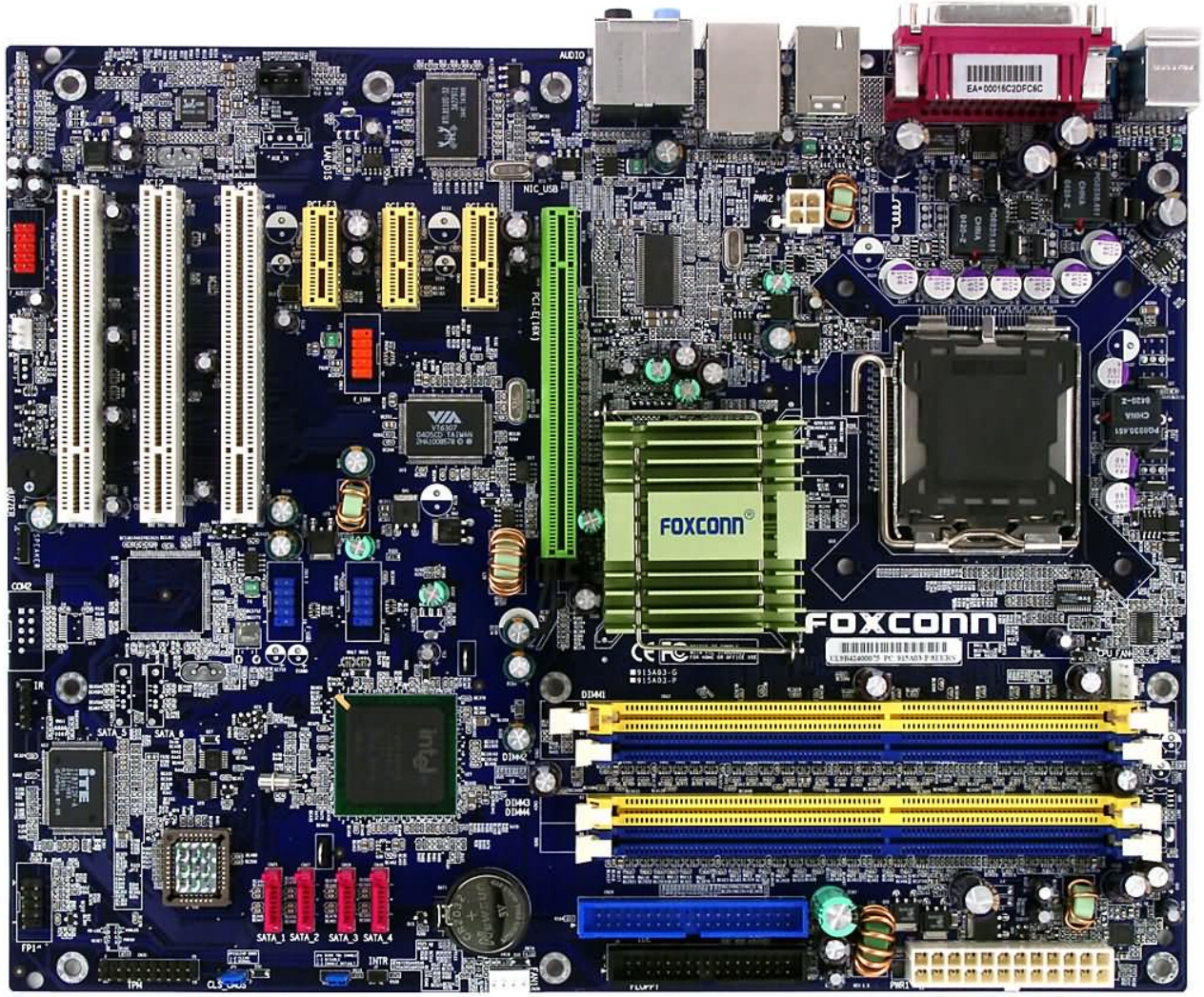
시스템 설치 과정

지금까지 설치 CD를 준비하고, 부팅하고, 디스크를 분할하여 마운팅을 했습니다. 지금까지는 하드디스크라는 '땅'에 구획을 정하고 용도에 맞게 땅을 고르는 작업을 한 셈입니다. 그러나 아직은 그 무엇하나 들어서지 않은 무주공산입니다. 드디어 이제부터 하나하나 젠투 리눅스를 위한 것들이 자리잡기 시작합니다. 차근차근 따라해

보시기 바랍니다.

시간 확인

혹시 시간이 맞지 않다면 시간을 맞추어줍니다. 아직은 대충 휴대전화의 시각을 참고해서 적당히 너무 차이가 나지 않게만 맞추어 주면 됩니다. 간혹 사용이 뜸했던 오래된 PC를 사용하다 보면 시계가 초기화되는 현상이 있습니다. 이것은 보통 메인보드에 내장된 하드웨어 시계를 돌려 주던 배터리가 방전되었기 때문입니다. 보통 메인보드를 살펴보면 단추 전지 하나씩은 달려 있습니다. 이것을 새것으로 교환하면 전원이 꺼져도 시계는 계속 동작합니다.



위 메인보드 그림 아래쪽 4개의 빨간 SATA 포트와 파란색 IDE 포트 사이에 동그란 단추 전지가 있는 것이 보이시나요?

시간을 맞추거나 확인하는 명령은 date입니다.

```
# date
Mon Jul 15 09:44:12 UTC 2013
```

여기서 UTC는 협정 세계시각임을 뜻합니다. GMT라고도 하지요. 한국 시각이면 KST라고 나올 겁니다. UTC에 9시간을 더하면 KST입니다. KST인지 UTC인지 확인하고, UTC로 출력이 되었다면 입력도 UTC에 맞추어 입력하세요. 만일 현재 서울의 시각, 즉 KST가 2013년 7월 17일 오후 7시 30분이라면, UTC는 2013년 7월 17일 오전 10시 30분입니다.

```
# date 071710302013 # MMDDhhmmYYYY
```

스테이지3 받기

비유로 설명하자면 사실 설치 CD에는 땅을 고르게 하기위한 기구들만 있었지, 그 땅에 건물을 세우기 위한 '기자재'들은 전혀 포함되어 있지 않았습니다. 이러한 기자재들은 이제 인터넷을 통해 다운로드 받아야 합니다. 이미 이전 단계에서 네트워크 설정을 잘 해 두었으니까, 조금은 불편하긴 해도 이 시스템으로도 충분히 원격지에서 파일을 가져올 수 있습니다.

기왕이면 가장 빠른 미러에서 파일을 가져오는 것이 좋습니다. 한국이라면 다음이나 카이스트가 가장 빠른 미러 서버일 겁니다. 그러면 다음 미러 서버에서 stage3 파일을 다운로드 받도록 합니다.

정말 투박하긴 해도 CLI에서 쓸 수 있는 웹브라우저들이 있습니다. 젠투 설치 CD에는 기본적으로 links라는 작은 텍스트 전용 웹브라우저가 내장되어 있습니다.

```
# cd /mnt/gentoo # 하드 디스크에 파일을 저장하도록 미리  /mnt/gentoo로 이동해둡니다.
# links http://ftp.daum.net/gentoo/releases/amd64/current-stage3
```

스플래시 화면 (혹은 welcome 화면)이 나오면 '엔터'를 눌러줍니다. 위/아래 방향키를 눌러 하이퍼링크 간 이동을 합니다. 참고로 좌/우 방향키는 이전/이후 페이지로 이동하는 기능입니다. ESC 키를 누르면 화면 상단에 메뉴가 나오므로 참고하면 됩니다. amd64 아키텍쳐를 이용할 것이므로 반드시 'stage3-amd64-20130711.tar.bz' 파일을 다운로드 받습니다. ISO 파일 때와 마찬가지로 파일의 무결성을 확인하는 것이 좋습니다. 이렇게 따라하시면 됩니다.

1. 'stage3-amd64-YYYYMMDD.tar.bz.DIGESTS' 파일을 받습니다. 받을 때 links가 멋대로 파일의 확장자를 바꾸는 일이 있으므로 다운로드 확장자를 바뀌지 않게 해서 다운로드 받습니다.
2. 다음 명령어로 확인하면 됩니다. # sha512sum -c stage3-amd64-<YYYYMMDD>.tar.bz2.DIGESTS
3. DIGESTS 파일을 보면 4개의 검증값이 있습니다. stage3-amd64-<YYYYMMDD>.tar.bz2에 대한 SHA512 HASH 값만 검사하면 되므로 첫번째 결과만 OK 사인이 떨어지면 되고, 나머지 3개에 대해서는 에러가 나거나 FAIL해도 아무런 문제가 없습니다.

이 stage3 파일은 기본적으로 필요한 툴체인들을 가지고 있습니다. 즉, 이 파일은 새로 만들어진 젠투 시스템이 최소한으로 필요한 '자가생산설비'들을 담고 있는 아주 중요한 파일입니다. 예전에는 지금보다 더 좋지 않은 하드웨어로도 stage1, stage2부터 시스템을 구축하는 '도전'들이 많이 있었는데, 한마디로 '미친 짓'입니다. 정말 리눅스를 바닥부터 만들어 보고 싶은데, 배포판의 편리함 또한 포기하는 싫어 젠투를 쓰겠다는 그런 '예외적인 케이스'가 아니라면 그냥 stage3를 쓰세요.

파일을 다 다운로드 받았다면 파일을 설치할 디스크에 풀어줍니다.

```
# tar xjpf stage-amd64-YYYYMMDD.tar.bz -C /mnt/gentoo
```

파일의 내용이 잘 기록되었는지 확인합니다.

```
# ls /mnt/gentoo
```

Portage 스냅샷 받기

소프트웨어 의존성과 패키지 관리

잠시 지금 우리가 하는 일이 어떤 작업인지 그 의미를 생각해 보도록 하지요. 소프트웨어들은 서로 유기적으로 구성되어 있습니다. 아주 단순한 프로그램이라면 독립적으로 구성되어 있을 수 있지만, 왠만큼 크기가 있는 프로그램들이라면 거의 모두 미리 만들어진 다른 '라이브러리(library)'를 필요로 합니다.

라이브러리란 어떠한 특정 기능들을 묶어 별도의 소프트웨어로 구성한 것을 말합니다. 예를 들어 어떤 프로그램에서 압축 기능이 필요로 한다고 하죠, 프로그래머가 직접 압축 기능을 프로그램을 만들어 낼 수도 있습니다. 그러나 아주 특수한 상황이 아니라면 압축만을 전문으로 담당하는 라이브러리를 가져다 쓰는 것이 훨씬 이득이 클 것입니다. 이것은 인테리어 도배 작업을 할 때 도배 전문가를 불러 작업하는 것과 같은 이치입니다. 직접 도배 도구와 벽지 등등을 산 다음 DIY 정신을 살려서 스스로의 손으로 작업을 할 수도 있지만, 도배란 것이 그리 쉬운 일이 아니지요. 전문가가 직접 하는 것이 훨씬 빠르고, 결과도 좋기 때문에 사서 고생을 하는 것보다는 보통 도배 전문가를 불러 작업을 맡기죠. 소프트웨어도 도배와 크게 다르지 않습니다. 라이브러리 개발자가 더욱 전문적으로 설계한 프로그램의 편리성과 정확성, 그리고 속도를 당해 내기는 어렵습니다.

거의 모든 소프트에어들은 기존의 훌륭한 라이브러리를 기반으로 하여 제작됩니다. 기존의 기능은 기존의 라이브러리를 사용하고, 새로운 기능은 새로운 프로그램에서 구현해내는 것입니다. 물론 새로이 만들어진 프로그램 또한 수많은 검증을 거친 다음에는 또 하나의 라이브러리로 인정받을 수 있습니다.

그러므로 어떤 프로그램은 다른 프로그램을 필요로 하는 현상이 매우 자주 발생하는데 이러한 현상을 일컬어 '의존성(dependency)'이라고 합니다. 다시 말해 “B 라이브러리(소프트웨어)는 A 라이브러리에 의존성을 가진다.” 란 말은 A라는 라이브러리(패키지)가 설치되어 있어야만 B 라이브러리(패키지)가 올바로 설치되고 동작된다는 뜻입니다.

portage

보통 소프트웨어 개발에서 이러한 의존성은 매우 골치아픈 문제입니다. a라는 프로그램을 설치하려 합니다. 그런데 이것은 b라는 라이브러리를 필요로 합니다. 그럼 b를 먼저 설치해야죠. 그런데 b는 또다른 라이브러리 c, d, e에 의존성이 있습니다. 그리고 각각의 c, d, e는... 진절머리나겠죠?

보통 리눅스 배포판은 이러한 프로그램의 카탈로그를 만들어 두고 있습니다. 이 카탈로그의 정보를 활용하면 복잡한 의존성 문제를 쉽게 해결할 수 있습니다. 미리 어떤 소프트웨어는 어떤 소프트웨어에 의존성을 가지는지, 이 소프트웨어는 어떤 성격을 가지는지, 어떻게 설치하는지 등의 자세한 정보를 별도로 관리하는 것이죠. 이 장치(소프트웨어)를 일컬어 '패키지 관리자'라 부릅니다. 특히 유명한 것이 데비안 리눅스 계열의 'apt'입니다. apt와 portage와의 차이점이라면 apt는 각 아키텍쳐마다 미리 컴파일된 바이너리를 받아 설치하는 반면, portage는 기본적으로 소스를 받아 직접 컴파일하는 형태를 취하는 점이겠지요.

현재 우리 시스템은 아주 기본적인 생산 설비만을 갖추어 둔 상태입니다. 시스템은 스스로 소프트웨어를 생산할 최소한의 준비만을 마친 상태이지요. 그러나 '어떤 것을 어떻게' 생산해야 할지에 대한 체계는 아직 갖춰지지 않았습니다. 그러므로 우리는 portage 시스템을 디스크에 설치함해 그러한 체계를 갖추도록 해야 합니다.

설치

이제 portage의 스냅샷 파일을 다운로드 받아 보겠습니다. Portage 스냅샷이란 젠투 리눅스가 관리하는 소프트웨어 목록을 압축한 파일입니다. 사실 이 작업은 나중에 make.conf 설정 및 chroot후 하게 되는 `emerge –sync` 라는 명령의 결과와 동일합니다. 그러나 이 관리 항목의 수가 상당히 많기 때문에 `emerge –sync`의 처리 과정보다는 압축해서 단일 파일로 한꺼번에 받아 PC에서 한번에 풀어주는 것이 더 신속합니다. 그러므로 설치할 때 기왕이면 빠른 방법이 낫다고 판단하여 문서에 기재하지만, 이러한 방법을 원치 않는다면 chroot 후에 `emerge –sync` 명령을 내려도 전혀 지장이 없습니다.

stage3를 받을 때와 동일하게 다음 미러 서버의 http://ftp.daum.net/gentoo/snapshots/ 로 접속합니다. 그리고 portage-latest.tar.bz 파일을 다운로드 받으면 됩니다. 참고로 이 스냅샷 파일은 CPU 종류와는 무관합니다. 파일의 전송이 끝나면 다음 명령으로 파일을 풀어줍니다.

```
# cd /mnt/gentoo # 하드디스크 (/dev/sda5)에 파일을 받도록 미리 처리해둡니다.
# tar xpjf portage-latest.tar.bz -C /mnt/gentoo/usr
```

portage 살펴보기

/mnt/gentoo/usr 디렉토리를 살펴보면 많은 디렉토리가 생성되어 있고, 그 디렉토리에 또 여러 디렉토리가 채차 생성된 것을 보실 수 있습니다. 이것이 portage tree입니다. 젠투 리눅스가 관리하는 프로그램을 목록으로 만들고, 비슷한 성격을 가지는 목록끼리 서로 묶어 카테고리화하고, 재차 세부적으로 카테고리화를 시켜 계층화한 것입니다. 일례로 `/mnt/gentoo/usr/portage/www-servers/apache` 디렉토리를 살펴보면 다음과 비슷한 파일들이 있는 것을 볼 수 있습니다.

```
# ls /usr/portage/www-servers/apache
ChangeLog       Manifest                apache-2.2.24.ebuild
apache-2.4.4.ebuild  metadata.xml
ChangeLog-2008  apache-2.2.24-r1.ebuild  apache-2.4.4-r3.ebuild  files
```

portage는 서버 프로그램들을 묶어 'www-servers'라는 카테고리로 1차 분류합니다. 서버 중에는 apache가 있지요. Apache는 여러 버전이 있는데, 현재는 2.2.24, 2.2.24-r1, 2.4.4, 2.4.4-r3 이렇게 네 가지 버전이 제공되고 있습니다.

make.conf

젠투의 가장 매력적인 부분을 꼽자면 이 make.conf입니다. 데비안, 페도라, 오픈수세와 같은 리눅스와는 조금 다르게 젠투는 필요한 프로그램을 직접 컴파일하는 배포판입니다. 기본적인 젠투의 철학이 사용자가 모든 사항을 선택하고 결정할 수 있도록 하는 것이라 그렇기도 하고, 애초에 BSD ports 시스템에 영향을 받은 배포판이 젠투입니다. 프로그램의 소스를 가져다 컴파일을 하기 때문에 사용자가 프로그램에 관해 세세하게 조정 가능합니다.

사용자가 시스템에 돌아가는 사항을 투명하게 파악할 수 있으므로, 이 점은 장점과 동시에 단점을 가지고 있습니다. 모든 것을 선택하고 조정할 수 있기 때문에 정말 자신의 취향에 맞게 시스템을 설계할 수 있다는 유연성, 그러면서도 상당히 그러한 시스템을 편리하게 만들어낼 수 있다는 점은 젠투라는 배포판이 가진 막강한

장점입니다. 그러나 그런 장점을 거저 얻을 수는 없지요. 사용자 또한 시스템에 잘 알고 있으며 능숙하게 다룰 수 있다는 전제 하에 가능한 이야기입니다. 그리고 시스템을 매우 간결하게 유지할 수 있는 장점이 있지만 데스크탑으로 PC를 활용할 때, 굳이 데비안과 비교를 하자면, 데비안이라고 해서 그렇게 패키지가 난잡하게 꾸려지는 것도 아니거니와 패키지 몇 개 더 깔린다고 해서 시스템에 치명적인 악영향을 주는 것도 아닙니다. 그저 하드디스크 용량을 더 먹는 정도인데, 대개 하드디스크는 충분히 크고도 남습니다.
또한 직접 컴파일해 패키지를 생산하는 것은 마치 '밖으로 비치는 주방을 갖춘 식당'과 비슷합니다. 만들어지는 과정을 일일이 볼 수 있다는 점이 장점이라면 분명 장점이지만, 요리를 모르면 백날 창을 통해 요리사가 무얼 하는지 이해하지 못하며, 그러므로 잘 모르면 별 의미가 없는 겁니다. 또한 사실 식당 요리의 위생과 맛은 식당 운영자들의 양심, 성실성, 그리고 실력의 문제이지, 주방이 훤히 보인다고 해서 요리가 더 위생적이 되거나 더 맛있는 음색이 탄생하는 것은 전혀 아닙니다.

데비안 또한 각 아키텍쳐에 맞게 일반적으로 가장 훌륭한 옵션을 만들어 컴파일을 할 것이라 생각합니다. 잘은 모르지만, 프로그램이 워낙 많고 아키텍쳐도 여럿이니 그에 맞춰 쿵떡쿵떡 찍어내긴 하겠죠. 그렇다고 설렁설렁 만들어 낼거라고는 생각하지 않습니다. 데비안의 미리 만들어진 패키지가 젠투에 비해 성능이 많이 떨어진다는 이야기는 들어본 적도 없고, 실제로 젠투의 패키지와 비교한 결과를 결과와 비교하면 근소한 차이가 날 뿐입니다 (젠투가 늘 낫다는 것도 아니란 뜻입니다). 우분투와 젠투를 벤치마크한 결과를 참고해보세요. 젠투를 사용하다고 해서 시스템이 획기적으로 빨라지는 것은 아닙니다. 다만 몇몇 설정들은 젠투가 가벼운 경우가 종종 있고, 그러므로 조금 더 쓰기에 쾌적해 질 수는 있습니다. 그렇다고 우분투는 그렇게 만들기 아주 불가능한가? 근본적으로 같은 리눅스를 쓰는데 불가능할리가요.

컴파일 옵션 설정

make.conf 파일은 패키지를 제작할 때 설정할 수 있는 여러 선택하고 조정할 수 있도록 하는 장치입니다. make.conf 파일은 `/mnt/gentoo/usr/portage` 에 있습니다. 이 파일을 열어 보면 기본적인 세팅은 되어 있습니다.

우선 CFLAGS 옵션을 조정해보도록 하겠습니다. 이것은 gcc 컴파일러에 전달할 컴파일 옵션입니다. 그러므로 각 시스템마다 조금씩 달라질 수도 있습니다. CFLAG의 관한 설명은 젠투 위키를 참고하고, 또한 안전한 CFLAG에 관한 위키도 참고하세요. 요즘은 `-march=native` 옵션이 있으므로 그렇게 큰 고민을 하지 않아도 되는 것 같습니다.

```
CFLAGS="-O2 -pipe -march=native"
```

CXXFLAGS는 건드리지 않아도 됩니다.

컴파일 옵션 문서를 참고하면 더 많은 정보를 얻을 수 있습니다.

CHOST

```
CHOST="x86_64-pc-linux-gnu"
```

이 값은 절대 변경하지 마세요.

MAKEOPTS

요즘은 CPU가 다들 멀티코어로 되어 있습니다. 즉 한 번에 동시에 여러 파일을 컴파일할 수 있죠. MAKEOPTS는 항상 사용하는 것이 유리합니다. 만일 CPU의 코어가 4개라면 아래처럼 적습니다.

```
MAKEOPTS="-j4"
```

자신의 CPU 코어 개수에 맞춰 적절히 숫자를 조정하세요.

USE Flag

패키지를 만들 때 필요한 프로파일입니다. Portage는 이 플래그들을 보고 각각의 패키지를 만듭니다. 프로파일을 명시적으로 더하려면 프로파일의 이름의 앞뒤를 공백으로 구분하여 적고, 프로파일을 명시적으로 빼려면 프로파일의 이름 앞에 '-'를 붙여 주면 됩니다. USE 플래그에 따라 같은 패키지라도 기능이 약간 달라질 수 있습니다.

처음 젠투를 설치한다면 이 부분에 시간을 조금 들여 자세하게 살펴 보는 것이 좋습니다. 모든 USE 플래그는 인터넷 웹페이지나 `/usr/portage/profiles/use.desc` 파일에 기술되어 있습니다[1]. 모두 외울 필요는 물론 없습니다만, 한 번 살펴보는 것은 도움이 됩니다.

일단 처음에는 `bindist` 플래그[2] 하나만 설정되어 있을 것입니다. 현재 제 플래그를 간단히 소개하겠습니다. 적절히 보고 참고하시면 됩니다. 시스템마다, 또 취향마다 USE 플래그의 조합은 다양해집니다. 이 부분은 각자의 연구가 필요합니다.

- 사용하지 않도록 만든 플래그 (모두 앞에 '-'를 붙입니다)
  - gnome 제가 최근 MATE('마떼', '메이트'가 아닙니다) 데스크탑을 즐겨 사용하기 때문에 GNOME을 저버린 상태입니다. 그래서 GNOME은 사용하지 않도록 막아 두었습니다.
  - ipv6 공유기 하에서 사용중이라 IPv6는 사용할 일이 없으므로 모든 패키지는 IPv6 기능을 해제한 채로 설치하도록 했습니다.
  - kde GNOME과 마찬가지로 저는 예전부터 KDE를 사용하지 않았습니다.
- 사용하도록 만든 플래그
  - X 데스크탑을 이용하면 X 윈도우를 사용해야겠지요. 그러므로 X는 명시하도록 합니다.
  - alsa 사운드와 관련된 플래그입니다.
  - cdr CD 레코딩과 관련된 플래그입니다.
  - cjk 몇몇 한중일 언어 관련에서 있을 수 있는 설정을 살리기 위해 사용했습니다.
  - dbus D-BUS 메시지 시스템을 사용하도록 명시했습니다. 대부분의 경우 명시하는 것이 좋습니다.
  - dvd DVD 드라이브가 장착된 PC를 사용하므로 패키지도 그에 맞춰 제작되는 것이 좋습니다.
  - gtk gtk 라이브러리를 사용하도록 합니다.
  - mmx CPU의 mmx 기능을 사용하도록 명시합니다.
  - qt4 QT4 라이브러리를 사용하도록 명시합니다.
  - sse
  - sse2
  - sse3 그래도 제 CPU가 아주 구형은 아닌지라 SSE, SSE2, SSE3와 같은 확장 명령어들을 지원합니다. 명시적으로 지원하도록 했습니다. `cat /proc/cpuinfo` 명령을 수행하면 CPU 정보를 자세하게 살펴볼 수 있습니다.
  - threads 스레드를 사용하도록 했습니다.
  - udev 장치와 관련된 많은 패키지가 이를 사용합니다.
  - vaapi
  - vdpau 제 그래픽 카드는 nVidia 계열입니다. 그래픽 카드로 동영상 가속이 가능한데, 위 둘은 이것을 가능하도록 명시한는 플래그입니다.

make.conf 파일에서 USE 플래그를 조절하면 시스템 전체에 영향을 미칩니다. 가끔씩 매우 중요한 플래그를 건드리면 시스템 전체에 평지풍파(정말입니다!)를 일으키기도 합니다. 그렇다고 플래그를 잘못 설정한다고 해서 시스템이 완전히 망가지거나, 엄청나게 비효율적으로 되는 일은 드물기 때문에 그렇게 크게 걱정하지 않아도 됩니다.
한편 설치할 패키지 단위로 세세하게 USE 플래그를 조절하는 방법도 있습니다. 이것은 차후 portage 관리에서 설명드리도록 하겠습니다.

USE Flag의 예

USE 플래그가 어떤 역할을 하는지 간단한 예를 들어 설명해 보도록 하겠습니다. 'app-arch/zip' 패키지는 파일들을 zip 포맷으로 압축할 때 필요하며 'app-arch/unzip' 패키지는 zip 파일을 압축 해제하기 위해 필요합니다. 한편 이 zip 파일을 사용해 압축을 하는 경우 약간의 결함이 발생하는 경우가 있습니다. 윈도우에서 압축한 한글 파일이 리눅스에서는 깨져 보이는 것입니다[3]. 이것은 문자 인코딩 문제인데, 가끔 웹페이지의 글자가 깨져 보인다거나, 동영상의 자막이 깨져 보이는 것과 근본적으로 동일한 원인 때문입니다. 윈도우에서는 자체적으로 CP 949라는 코드페이지를 사용하는데, 리눅스는 거의 UTF-8 인코딩을 사용하거든요. 둘의 문자를 처리하는 방식이 서로 다르고, zip은 이러한 인코딩 방식의 차이에 대해 기본적으로는 처리하지 않게 되어 있었습니다[4]. 그렇다고 해서 zip 파일로 압축을 풀면 무조건 파일 이름이 엉망이 될 수 밖에 없는가? 그렇지는 않습니다. zip, unzip 명령을 사용할 때 압축된 파일의 인코딩 방식을 알면 올바르게 파일 이름을 복구할 수 있도록 하는 옵션을 가지고 있습니다.

```
# unzip a.zip # 파일 이름이 완전히 깨진다
???????.txt
???????.smi
# unzip -O cp949 a.zip # 파일 이름이 잘 복구된다
테스트파일.txt
테스트자막.smi
```

위 예와 같이 '-O <codepage>' 옵션을 사용하면 파일 이름이 깨지는 현상은 극복할 수 있습니다. 리눅스를 꽤 오래 사용하셨던 분들이라면 한 번쯤은 경험해보셨을 문제이리라 생각합니다. 그런데 젠투에서 기본적으로 만들어지는 unzip/zip을 사용하면 '-O' 옵션을 알아듣지 못하고 에러를 냅니다. 어떻게 된 것일까요?
그것은 zip과 unzip을 만들 때 USE 플래그에 '-O' 옵션을 사용하도록 하는 프로파일을 집어넣지 않았기 때문입니다. '-O' 옵션은 natspec이란 USE flag가 있어야 합니다.

나중에 젠투 시스템이 완전히 설치된 후, 다음과 같은 명령어로 패키지를 설치한 후 실습해 보세요

```
# emerge -av gentoolkit  # gentoolkit 패키지를 설치합니다.
# equery uses unzip      # unzip 패키지가 사용하는 USE 플래그의 목록과 설명을 확인합니다.
[ Legend : U - final flag setting for installation]
[        : I - package is installed with flag     ]
[ Colors : set, unset                             ]
 * Found these USE flags for app-arch/unzip-6.0-r3:
 U I
 + + bzip2   : Use the bzlib compression library
 + + natspec : Use dev-libs/libnatspec to correctly decode non-ascii file
names
               archived in Windows.
 + + unicode : Adds support for Unicode
```

natspec 플래그는 윈도우에서 아스키 파일이 아닌 파일 이름을 올바르게 디코딩할 때 필요한 라이브러리를 같이 사용하라는 뜻이군요. 정말 완전히 영미권의 사람이라면 굳이 필요하지는 않은 옵션입니다. 본 예와 같이 젠투는 필요한 기능을 USE flag를 통해 유연하게 조절할 수 있다는 장점이 있습니다.

ACCEPT_KEYWORDS

64비트 CPU를 사용하는 경우 패키지 설치 때 자주 ~amd64 키워드를 허용해 달라는 메시지를 볼 수 있습니다. 일일이 이 키워드를 설정하기 귀찮으니 처음부터 시스템 전역으로 ~amd64 키워드는 설정해두었습니다.

```
ACCEPT_KEYWORDS="~amd64"
```

이 말은 어떤 패키지들에는 ~amd64라는 키워드가 붙습니다. 이 말의 의미는[5] 해당 패키지들은 64비트 CPU에서 충분히 검증되지 않은 시험 단계의 패키지라는 뜻입니다. 물론 크게 문제가 되면 사용하지 않는 것이 좋지만, 심각하게 문제가 되는 것은 아니므로 괜찮습니다.

GENTOO_MIRRORS

스테이지 3와 포티지 스냅샷을 통해 이미 미러 서버에 접속해 보았습니다. 앞으로 항상 패키지 소스는 미러 서버로부터 받기를 원한다는 것을 명시하기 위해 사용합니다.

```
GENTOO_MIRRORS="http://ftp.daum.net/gentoo/"
```

다음이 아닌 미러를 사용하고 싶다면 GENTOO_MIRRORS 항목을 제거하고 다음 명령어를 통해 미러를 선택하도록 합니다.

```
mirrorselect -i -o >> /mnt/gentoo/etc/portage/make.conf # '>' 기호는 항상 2개여야 함에 주의합니다.
```

LINGUAS

어떤 패키지들은 다국어를 지원하는데 (예: 리브레오피스) 설치할 때 다국어 지원을 허용하려 할 때 필요한 변수입니다.

```
LINGUAS="ko" # 한국어 지원을 허용합니다.
```

X를 위한 설정

거의 모든 이가 X윈도우 환경을 사용할 것이므로 두 항목도 설정합니다.

```
INPUT_DEVICES="evdev"
```

```
VIDEO_CARDS="nvidia"
```

만일 노트북을 사용하고, 터치패드를 추가로 사용하기를 원한다면 `INPUT_DEVICES` 항목에 `synaptics` 값을 추가해주세요. 그리고 저는 nvidia의 독점 드라이버를 사용하기 때문에 nvidia를 선택했습니다. 만일 nVidia 카드를 사용하면서 오픈 소스 드라이버인 nouveau를 사용하고 싶다면, `nvidia` 값을 삭제하고 `nouveau` 로 변경하면 됩니다. 만일 Radeon 계열의 그래픽 카드를 사용한다면 이 값을 `radeon`(오픈소스), 혹은 `fglrx`(독점)으로 설정합니다. 인텔 그래픽 카드면 `intel`을 설정합니다.

나머지

```
source "/var/lib/layman/make.conf"
```

이 값은 차후 Gentoo overlay를 사용할 때 필요합니다. 일단 적어두고 주석화시켜 둡니다.

시스템 설정

Chroot 준비

make.conf의 GENTOO_MIRRORS가 올바로 되어 있는지 다시 확인합니다.

DNS 정보를 복사하고, 나머지 시스템에 필요한 파일시스템을 마운트합니다.

```
# cp -L /etc/resolv.conf /mnt/gentoo/etc/
# mount -t proc none /mnt/gentoo/proc
# mount --rbind /sys /mnt/gentoo/sys
# mount --rbind /dev /mnt/gentoo/dev
```

Chroot

이제 하드디스크의 시스템으로 진입합니다.

```
# chroot /mnt/gentoo /bin/bash
# sourcec /etc/profile
# export PS1="(chroot) $PS1"
```

이제 여러분은 새롭게 만든 PC의 젠투 시스템으로 접근한 것입니다. 디렉토리의 루트('/')는 더이상 설치 CD의 그것이 아닌, 우리가 지금까지 작업한 /dev/sda5, 즉 /mnt/gentoo가 새롭게 디렉토리의 루트가 된 것입니다. 설치가 다 끝나고 PC의 하드디스크로 부팅을 하게 되면 보게 될 화면을 미리 만나보고 계신 것입니다.

Portage 설정

이전 단계에서 snapshot 파일을 받아 압축을 /mnt/gentoo/usr에 풀었다면 이 작업은 무시해도 됩니다. 사실상 같은 내용입니다.

```
# emerge --sync
```

Profile 설정

프로파일은 USE나 CFLAGS 말고도 시스템에 기본적으로 필요한 여러 설정을 미리 정해둔 프리셋입니다.

```
# eselect profile list
Available profile symlink targets:
  [1]   default/linux/amd64/13.0
  [2]   default/linux/amd64/13.0/selinux
  [3]   default/linux/amd64/13.0/desktop *
  [4]   default/linux/amd64/13.0/desktop/gnome
  [5]   default/linux/amd64/13.0/desktop/kde
  [6]   default/linux/amd64/13.0/developer
  [7]   default/linux/amd64/13.0/no-multilib
  [8]   default/linux/amd64/13.0/x32
  [9]   hardened/linux/amd64
  [10]  hardened/linux/amd64/selinux
  [11]  hardened/linux/amd64/no-multilib
  [12]  hardened/linux/amd64/no-multilib/selinux
  [13]  hardened/linux/amd64/x32
  [14]  hardened/linux/uclibc/amd64
```

저는 일반적인 데스크탑 환경인 3번을 선택한 상태입니다. 원하시는 환경에 맞춰 profile을 선택하면 됩니다. 데스크탑 환경으로 GNOME을 사용할 예정이라면 4번을, KDE을 사용할 예정이라면 5번을 선택하고, 나머지는 저처럼 3번을 선택하면 됩니다. 서버라면 1번을 선택하세요. 2번과 9번 이후는 SELinux들인데, 일반적인 환경에서는 굳이 사용하지 않아도 됩니다. 일반적인 용도라면 1, 3, 5, 6번중 하나를 고르면 됩니다. 그리고 이 프로파일 리스트는 때때로 변화할 수 있습니다. 적절히 프로파일의 이름을 보고 용도와 가장 잘 맞는 것을 하나 골라 선택하시면 됩니다.

```
# eselect profile set 3
```

시간대 설정

한국에 거주하시는 분이라면 Asia/Seoul, KST (UTC+9)을 선택할 것입니다.

```
# cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime
# echo "Asia/Seoul" > /etc/timezone"
```

커널 설정

이제 기본적인 시스템 세팅이 끝났습니다. 이제부터는 emerge를 통해 필요한 패키지를 설치하는 것도 가능하게 되었습니다. 그러나 아직 커널과 부트로더가 설치되지 않았기 때문에 PC를 재부팅하게 되어도 현재 시스템으로 들어오는 것은 아직 불가능합니다. 그러므로 이 장에서는 커널 컴파일에 대해 간단하게 다루고 넘어가겠습니다.

커널 컴파일은 수동으로 하거나, 젠투의 genkernel을 사용하는 방법 둘 중의 하나를 선택할 수 있습니다. 본 문서에서는 'genkernel'을 사용하는 과정만을 다루겠습니다. 일단 genkernel 사용하는 방법을 터득했으면 수동 컴파일 문서를 참고해서 따라할 수 있을 것입니다.

커널

일단 '커널(kernel)'이란 것이 무엇인가요? 'Kernel'의 사전적인 의미는 '알맹이'입니다. 운영체제(OS, Operating System)의 핵심이 되는 알맹이 부분이란 뜻이죠.

우리 눈에 보이지는 않지만 커널은 CPU, 메모리, 하드디스크와 같은 장치들을 제어하고, 프로그램의 실행을 담당하는 역할을 합니다. 커널은 컴퓨터를 살아 움직이게 하는 심장이나 두뇌 같은 핵심적인 소프트웨어입니다. 운영체제에서 커널이 없다면 아무 것도 할 수 없습니다. 이렇게 중추적인 역할을 담당하므로 리눅스의 커널은 상당히 방대하며 갖가지 컴퓨터 기술들이 교차되는 분야이기도 합니다. 어떤 하드웨어든, 소프트웨어이든 결국 컴퓨터가 그것을 제어하려면 OS가 필요하고, OS의 핵심은 커널이니까요.

이 장에서 우리는 amd64 아키텍쳐를 위한 리눅스 커널을 만들 것입니다. 커널에 대한 설명은 워낙 방대하고 복잡하므로 본 문서에서 모든 것을 커버할 수 없습니다. 컴파일을 위한 최소한의 사항만을 언급하고 넘어가겠습니다.

커널 컴파일

우선 커널 소스와 genkernel 패키지를 받습니다.

```
# emerge gentoo-sources genkernel
```

아주 쉬운 방법: genkernel all

일단은 처음은 안정적으로 PC를 구동시키는 것이 좋습니다. 처음 젠투를 설치하시는 분들이라면 처음부터 커널 옵션을 건드려 일을 힘들게 만들 필요가 전혀 없습니다. 일단 거의 모든 하드웨어와 파일시스템 거의 모든 설정이 포함된 채로 커널을 만들어 봅니다. 일단 안정적인 가장 초기의 커널을 컴파일 한 후, 차츰차츰 자신의 시스템에 필요없는 기능을 빼가면서 커널을 가볍게 만드는 것이 좋습니다.

```
# genkernel all
```

genkernel은 대부분의 시스템에서 매우 잘 돌아가게 되어 있지만, 시스템에 불필요한 기능도 상당히 많이 포함되어 있습니다. 여러가지 기능을 제거해서 자신의 하드웨어에 최적화된 커널은 `genkernel all`로 만들어낸 커널보다 확실히 가볍습니다. 요즘은 CPU가 워낙 빠르고 좋기 때문에 시간이 많이 줄기는 했습니다만,

그래도 커널의 컴파일에는 시간이 많이 걸립니다. 컴파일되는 동안 잠시 휴식을 취하세요

genkernel의 장점은 커널이 컴파일되고 난 후 나머지 작업을 알아서 해 준다는 것입니다.

1. 컴파일된 커널을 알아서 /boot 디렉토리에 카피해줍니다.
2. 카피된 커널의 이름을 알아서 변경해줍니다.
3. initramfs을 생성합니다.
4. /boot 디렉토리에 커널 컴파일에 썼던 설정값도 같이 복사해둡니다.

genkernel이 끝나면 파일이 /boot에 올바로 설치되었는지 반드시 확인합니다.

```
# ls /boot/kernel* /boot/initramfs*
```

입맛에 맞는 커널 설정: genkernel --menuconfig

genkernel all로 커널을 만들면 별의별 기능들이 다 활성화된 채로 커널이 만들어집니다. 좀 더 최적화를 원하시는 분들은 차츰차츰 커널을 최적화시켜가면 됩니다. 그런데 커널은 워낙 옵션도 많고, 또 최신 기술들이 금방금방 집적되는 부분인데다가, 방대하기까지 합니다. 문서 하나로는 수많은 커널의 옵션들을 일일이 설명하기란 불가능합니다. 조금은 무책임하지만, 이 부분은 사용하시는 분들의 역량에 맡기겠습니다.

최소한의 팁을 드리자면,

1. http://how-to.wikia.com/wiki/How_to_configure_the_Linux_kernel 같은 온라인 문서, 서적등을 많이 검색하세요.
2. 커널은 안정적으로 유지하는 것이 먼저입니다. menuconfig의 'save'와 'load'를 이용하면 커널의 설정값을 온전히 별도의 파일로 보관하는 것이 가능합니다. 안정적인 커널의 설정값을 만든 후 차츰차츰 값을 빼가면서 최적화하시면 됩니다.
3. genkernel로 커널을 생성할 때 컴파일 이후의 작업은 자동으로 이루어집니다. 같은 버전에 대해 여러번 genkernel을 실행하면 이전에 /boot에 있던 커널은 새로 컴파일된 커널에 덮어씌워질 염려가 있습니다. 의도치 않게 커널이 덮어씌워지지 않도록 주의하세요

genkernel이 끝나면 파일이 /boot에 올바로 설치되었는지 반드시 확인합니다.

```
# ls /boot/kernel* /boot/initramfs*
```

부팅 때 필요한 모듈 로드하기

우선 어떤 모듈들이 설치되었는지 확인해봅니다.

```
# find /lib/modules/<kernel version>/ -type f -iname '*.o' -or -iname '*.ko'
| less
```

이전 단계인 '자신의 PC 사양을 확인하기'에서 설치 CD가 자동으로 로드한 커널의 모듈의 목록을 확보해 두었습니다. 사실 커널이 알아서 필요할 때 모듈을 로드하도록 관리를 해 주니까 그 모듈의 목록 전부를 자동으로 로드하도록 만들 필요는 없습니다. 그러나 특정 모듈, 이를테면 그래픽 카드나 이더넷 카드 정도는 부팅하면서 커널이 명시적으로 로드하도록 해둘 필요는 있습니다.

일례로 저의 PC는 nVidia의 독점 그래픽 카드를 사용하도록 설정해 두었습니다. 그 모듈의 이름은 nvidia입니다. 그리고 저의 메인보드에 내장된 이더넷 카드는 r8169라는 커널 모듈을 필요로 합니다. 적어도 이 둘은 부팅하면서 커널이 자동으로 로드하도록 하는 것이 좋습니다[6)]

```
# nano -w /etc/conf.d/modules
modules_3_8_10="nvidia r8169" # Kernel 3.8.10이 시작할 때 자동으로 nvidia, r8169를 로드한다.
```

불필요한 모듈 로드를 막기

그리고 어떤 모듈은 로드가 되지 않도록 막아두어야 할 경우가 생깁니다. 예를 들어 nvidia 독점 모듈을 사용하면 오픈 소스 드라이버인 nouveau 드라이버는 로드되어서는 안 됩니다. 이렇게 커널이 명시적으로 특정 모듈을 로드하지 못하게 막기 위해서는 /etc/modprobe.d에 blacklist.conf 파일을 생성하고 그 곳에 불필요한 모듈의 이름을 적어주면 됩니다.

```
# nano -w /etc/modprobe.d/blacklist.conf
blacklist nouveau # nouveau 오픈소스 드라이버를 로드하지 않음
```

부트로더와 나머지 작업

시스템을 위한 커널을 처음으로 컴파일했다면, 이 커널이 시스템에 올바르게 동작할지 아직 보증할 수 없습니다. 그러나 잠깐, 아직은 재부팅하기 이릅니다! 우선은 '부트로더(bootloader)'를 설치하고 'fstab'이란 중요한 설정이 아직 남아있기 때문입니다.

부트로더

부트로더란 것은 간단히 말해, 시스템의 메모리에 커널을 적재하기 위해 필요한 프로그램입니다. 모든 프로그램은 실행되기 위해선는 램(RAM)이라는 메모리에 우선 적재되어야 합니다. 그렇지 않으면 실행될 수가 없습니다. 커널도 마찬가지입니다. 운영체제에서 커널의 가지는 중요성은 이전 장을 통해서 피력했습니. 그러나 커널이 아무리 중요한 프로그램이라도 메모리에 적재되지 않은 채로 실행될 수도 있는 특권을 지닌 녀석도 아닙니다. 또한 스스로 살아 움직이는 생물도 아닌데 알아서 메모리에 적재될 리 없지요. 그러므로 어떤 프로그램이 먼저 메모리에 로드된 다음, 이 프로그램이 커널을 메모리에 적재될 수 있도록 해야 합니다.

상당히 이상한 말이죠? 커널을 메모리에 올리기 위해서는 다른 프로그램이 메모리에 올라가 있어야 하고, ... 그럼 그 '다른 프로그램'은 어떻게 메모리에 올라는 걸까요? 그 프로그램 또한 살아 움직이는 생물체도 아닌데 말이죠.

이것은 메인보드 자체에 내장된 BIOS(Basic Input Output System)에 의해 가능합니다. 'BIOS 메뉴'라고 해서 부팅할 때 'del' 키를 연타해서 들어가려고 하던 그 BIOS 메뉴의 BIOS가 맞습니다. 이것은 메인보드를 살 때 메인보드에 딸려 있는, 메인보드의 ROM에 내장된 프로그램입니다. 예전에는 정말 단순한 역할만 했었지만 요즘은 별별 기능을 다 가지고 있긴 하지요.

아무튼 이 BIOS는 하드 디스크의 MBR(Master Boot Record)를 읽어들입니다. 보통 첫 섹터가 MBR인데 이 MBR에는 한 컴퓨터의 시스템이 최초로 구동할 때 필요한 중요한 정보들이 담겨 있습니다. 이 MBR에서 부트로더를 재차 읽어들입니다.

MBR이 읽어들인 부트로더는 램에 로드되고 커널을 로드하기 위한 준비를 마치고 사용자의 입력을 기다립니다. 한 시스템에 꼭 하나의 커널이 있는 것은 아닙니다. 젠투, 우분투, FreeBSD나 윈도우 등등이 설치되어 있을 수 있습니다. 부트로더는 그런 여러 OS 커널을 사용자가 유연하게 선택할 수 있도록 도와줍니다. 한편 각각의 커널 또한 초기 구동 때 여러 옵션 값들을 가집니다. 그러한 옵션들은 사용자가 그래도 부팅을 할 때 쉽게 변화할 수 있도록 해 주어야 합니다. 부트로더는 커널에 그러한 옵션 값을 전달하는 역할도 수행합니다. 부트로더가 커널을 성공적으로 읽어들이면 그 때 부터는 커널이 시스템의 전권을 잡고 시스템을 제어하기 시작합니다. 어떤 시스템이 동작 중이라면 커널은 메모리에 어딘가에 상주하고 있다는 뜻입니다.

커널 패닉

잠시 '커널 패닉'에 대해 언급하겠습니다. 부트로더에서 커널 패닉을 설명하려니 뜬금없긴 하지만, 부팅할 때 커널 패닉이 종종 발생니, 이곳에 설명을 적는 것이 나아 보입니다.

간혹 커널에 심각한 오류가 있는 경우에는 커널이 시스템 상에서 동작하기를 포기하고 멈추어 버립니다. 이러한 현상은 부팅 중에도 일어날 수 있고, 프로그램을 잘 사용하다가 갑자기 일어나는 경우도 있습니다. 이런 현상을 일컬어 '커널 패닉(kernel panic)'이라고 합니다. 컴퓨터 자체가 멈추어 버리기 때문에 상당히 심각한 오류에 해당합니다.

앞서 말씀드렸듯이 커널은 한 시스템의 심장이자 시스템의 가장 기저에서 동작하는 프로그램입니다. 커널의 에러는 그냥 일반 프로그램의 에러와는 비교할 수 없습니다. 일반 프로그램에서 오류가 나면 커널이 이를 알아채어 오류에 대처할 방법을 만들면 시스템은 계속 동작할 수 있는 여지가 있습니다. 그러나 커널의 오류는 그렇지 않습니다.
어쩌다 갑자기 게임 중에 게임이 강제로 멈추는 일이 발생했다고 가정하지요. 커널이 지속할 수 없을 정도로 심각한 어떤 오류를 발견하고 강제로 프로그램을 종료시킨 것입니다. 매우 짜증이 나긴 하겠지만, 그래도 컴퓨터 자체가 멈추지는 않지요. 그러나 시스템의 가장 밑바닥에 있는 '최후의 보루'인 커널에서 에러가 나면 누가 그 에러를 보살펴줄까요? 커널이 커널 스스로를 보살필 수도 있겠지만, 그것이 불가능할 만큼 심각한 오류라면? 네, 그것이 바로 윈도우의 블루 스크린입니다. 컴퓨터 시스템 입장에서는 이러한 커널 패닉은 큰 사고입니다. 이런 오류에는 결국 방법이 없습니다. 시스템을 재기동시키는 것만이 답이지요.

부트로더 설치

부트로더는 GRUB (GRand Unified Bootloader)을 사용하겠습니다. 사실 저는 LiLo (Linux Loader)를 단 한차례도 사용한 바가 없습니다. GRUB2가 나온지 몇 년이 되었고, 대부분의 배포판은 이미 GRUB2를 사용하고 있습니다. 그렇지만 젠투는 기본적으로는 아직 GRUB 구버전을 사용합니다. 젠투가 GRUB2를 지원하지 않아서? 그런 건 아닙니다. GRUB2가 구버전인 GRUB에 비해 기능이 더욱 향상되긴 했지만 기본적으로 젠투 리눅스는 사용자가 모든 설정 파일을 건드리고 수동으로 조작하는 방식을 취합니다. 그런데 GRUB2는 좀 더 기능적으로는 나을지 몰라도 사용자가 직관적으로 편집하고 설정하기에는 조금 번잡한 감이 있습니다. 물론 GRUB2를 원하면 언제든지 GRUB2를 이용해 부팅을 할 수 있습니다만, 여전히 공식 문서에서도 기존의 GRUB을 사용하므로 이 문서도 옛 버전의 GRUB을 사용하는 것으로 하겠습니다.

```
# emerge grub
# grep -v rootfs /proc/mounts > /etc/mtab
# grub-install --no-floppy /dev/sda
```

이제 /boot/grub/grub.cfg 파일을 열어 부팅 옵션을 적어 보도록 하겠습니다.

```
# Which listing to boot as default. 0 is the first, 1 the second etc.
```

```
default 0
# How many seconds to wait before the default listing is booted.
timeout 5
# Nice, fat splash-image to spice things up :)
# Comment out if you don't have a graphics card installed
splashimage=(hd0,0)/boot/grub/splash.xpm.gz

title Gentoo Linux 3.8.13 (genkernel)
root (hd0,0)
kernel /boot/kernel-genkernel-x86_64-3.8.13-gentoo root=/dev/ram0
real_root=/dev/sda5
initrd /boot/initramfs-genkernel-x86_64-3.8.13-gentoo

# The next four lines are only if you dualboot with a Windows system.
# In this case, Windows is hosted on /dev/sdb1.
title Windows XP
rootnoverify (hd1,0)
makeactive
chainloader +1
```

default

부트로더가 실행되었을 때 어떤 항목이 기본적으로 선택되어 있을지를 정하는 옵션입니다. 0이면 가장 첫번째를 선택하도록 합니다.

timeout

디폴트 항목으로 정해진 시간 이후 자동으로 부팅하도록 합니다. 적당히 시간을 주도록 합니다.

리눅스 부팅

리눅스르 부팅하려면 보통 다음 세 줄을 입력합니다.

- title: 부트로더에 표시될 이름입니다. 적절히 알아보기 편하게 적어 주면 됩니다. 보통은 리눅스 배포판의 이름과 커널 버전을 적는 것이 관례입니다.
- root: 부트 파티션의 위치입니다. 우리는 부트 파티션을 `/dev/sda1`에 위치시켰으므로 `(hd0,0)`으로 입력합니다. 중요한 사항입니다. `(hd0,0)` 문자열에는 공백을 포함시키지 말고 붙여서 써야 합니다.
- kernel: 커널의 위치입니다. 커널의 경로를 정확히 적어 주어야합니다. 한 글자라도 틀리면 안 됩니다. 주의 깊게 체크하세요.
  - root=/dev/ram0: initramfs를 사용하면 이렇게 옵션을 줍니다.
  - real_root=/dev/sda5: 실제 디스크에서 루트 파티션을 가리킵니다.
- initrd: initramfs를 사용하는 경우 지정합니다. genkernel은 기본적으로 initramfs를 사용합니다.

윈도우 부팅

GRUB은 리눅스 부팅 뿐 아니라, 다른 OS의 부팅도 가능하도록 해 줍니다. 여러분들의 PC에 윈도우가 별도로 설치되어 있고, 윈도우로 부팅하고 싶다면 다음과 같이 grub.cfg 파일에 적으면 됩니다.

- title Windows 7: 윈도우 이름입니다.
- rootnoverify (hd1,0): 윈도우 7의 부트로더는 /dev/sdb1 (윈도우 설치 화면에서 예약된 파티션으로 나오는 100 MB 남짓의 그것입니다.)라고 가정합니다.
- makeactive
- chainloader +1: 이렇게 하면 GRUB이 윈도우의 부트로더를 다시 불러들입니다. 그래서 윈도우 커널이 시스템의 메모리에 적재되고, 윈도우로 부팅할 수 있게 되지요.

사족: 롤링 릴리즈와 젠투

설명을 덧붙이자면 젠투 리눅스는 '롤링 릴리즈(rolling release)' 체제를 취하고 있습니다. 예를 들자면 이렇습니다. 우분투는 특정 버전에 따라 배포판의 시스템을 각기 새롭게 제작합니다. 그리고 각 버전마다 지원 기간이 있어 지원 기간이 종료되고 나면 싫어도 새로운 버전으로 업그레이드하거나 완전히 시스템을 새로운 버전으로 설치해야만 캐노니컬(우분투를 관리/지원하는 업체)의 지원도 받고, 시스템도 최신으로 유지할 수 있습니다. 반면 젠투는 아무리 오래된 상태라도 `emerge –sync`후 업데이트를 계속 하기만 하면 최신 젠투 버전이 됩니다. 즉, '배포판의 버전 업그레이드'라는 명시적인 절차가 존재하지 않습니다.

이런 롤링 릴리즈를 채택한 대표적인 리눅스 배포판은 젠투와 아치(arch) 리눅스입니다. 아치 리눅스는 리눅스 배포판 중에서도 업데이트가 빠르기로 유명합니다. 롤링 릴리즈가 업데이트만 계속하면 업그레이드라는 명시적 절차 없이도 최신 버전을 유지할 수 있기 때문에 빠른 업데이트에 유리한 구조이긴 합니다. 그러면 젠투 또한 롤링 릴리즈라 업데이트 속도는 빠른 편이지 않을까 생각할 수 있는데, 그렇지는 않습니다. 오히려 젠투는 업데이트가 살짝 느린 편에 속합니다. 보통 배포판 중에서도 안정성을 중요시하는 쪽은 업데이트가 느린 편이고, 안정성 보다는 최신 기술의 도입을 우선시하는 배포판은 소프트웨어의 업데이트 속도가 빠른 편입니다. 젠투는 롤링 릴리즈이지만 안정성을 보다 우선시하기 때문에 업데이트가 아주 빠른 편은 어닙니다.

fstab 조절

이제 거의 막바지입니다. 재부팅하여 시스템이 올바르게 확인하고픈 마음 이해합니다만, 아직 나머지 설정 하나가 빠져 있습니다. fstab이란 파일인데, 매우 중요합니다. 커널은 어떤 파티션을 어떤 디렉토리에 마운트해야 할지 결정해야 합니다. 이것을 지정해둔 파일이 /etc/fstab입니다.

```
# nano -w /etc/fstab

/dev/sda1    /boot      ext4    defaults,noatime   0 2
/dev/sda2    none       swap    sw                 0 0
/dev/sda5    /          ext4    dev,noatime        0 1
/dev/sda6    /home      ext4    dev,noatime        0 2
/dev/cdrom   /mnt/cdrom auto    noauto             0 0
```

1. 첫번째 열은 마운트할 파티션입니다.
2. 두번째 열은 마운트될 포인트입니다.
3. 세번째 열은 파티션의 파일시스템입니다.

4. 네번째는 마운트 옵션입니다. mount(8)나 아치 위키를 참고하세요
5. 다섯번째는 dump/pass입니다.
    1. 백업할 때 덤프할지 말지 정합니다. dump를 따로 설치하지 않으므로 모두 0입니다.
    2. fsck가 파일시스템을 검사할지, 또 검사한다면 검사의 우선순위를 결정합니다.
        1. 0: 검사하지 않습니다.
        2. 1: 검사합니다. 높은 우선권을 갖즙니다. 루트 파티션만 1을 갖습니다.
        3. 2: 낮은 우선권을 갖습니다. 루트 파티션 외에는 2를 줍니다.

/etc/fstab 파일에 오류가 없는지 다시 한번 더 체크하고 저장하세요.

마무리 작업

루트 패스워드 설정

아직 루트의 패스워드를 설정하지 않았습니다. 반드시 루트 패스워드를 초기화해주어야 합니다. 다음 명령어를 실행하여 루트의 패스워드를 설정해주세요. 당연히 잊어버리시면 안 됩니다!

```
# passwd
```

언마운트 및 정리

chroot 상태를 벗어나 원래 설치 CD의 경로로 돌아가겠습니다.

```
# exit
# cd ..
# umount -l /mnt/gentoo/dev/{shm,pts,}
# umount -l /mnt/gentoo/{boot,proc}
```

재부팅해보기

이제 재부팅을 하겠습니다. 모든 설치 과정이 깨끗하게 잘 되었다면 올바르게 젠투 설치 화면으로 들어갈 수 있을 것입니다.

```
# reboot
```

그러나 항상 실수는 있는 법입니다. 만일 어디선가 실수가 있었다면 제대로 부팅되지 않을 것입니다. 몇 가지 체크리스트를 제안하겠습니다. 처음부터 깨끗하게 부팅되지 않는다고 해서 너무 실망하지 마세요 어떤 단계에서 문제가 있었는지 파악해서 수정하면 됩니다.

- GRUB의 설치를 하지 않았을 수도 있습니다.
- GRUB의 에러 메시지를 보았다면, 그 에러 메시지를 노트해서 검색해보도록 합니다.
    - 시스템의 하드디스크가 여럿이라면 fdisk -l의 디스크 순서, 즉 /dev/sda, /dev/sdb의 순서가 BIOS

화면에서 설정된 디스크 순서와 같은지 확인해 보시기 바랍니다. BIOS는 디스크 순서를 바꿀 수 있는 기능을 가지고 있는데, 리눅스는 이를 허용하지 않을 수 있습니다.
  - (hd0,0)과 같이 디스크 드라이브 지정 문자에 오타가 있는 경우가 있을 수 있습니다. (hdX,Y)에 **공백이 없고 0부터 시작한다는 것**을 다시 한 번 강조합니다.
  - GRUB의 OS 선택 메뉴는 정상적이나 부팅이 되지 않는다면 커널 및 initrd에 파일 이름 및 커널 옵션에 오타가 있는 것입니다. 주의 깊게 설정된 값을 재확인하세요.
- 커널의 문제
  - 뭔가 부팅이 되는 듯하다가 커널 패닉, 어떤 이상한 메시지를 남기고 더이상 진행이 되지 않을 때 - 이것이 커널 패닉입니다. 아마 커널 옵션 `root`, `real_root`의 값에 오류가 있는 경우일 수 있습니다. 그리고 커널의 설정을 따로 조절했을 경우 필요한 모듈이 로드되지 않아서 발생할 수 있습니다.

PC에 설치된 젠투 시스템에 문제가 있어 제대로 부팅이 되지 않았다면 설치 CD를 이용해 시스템으로 들어갈 수 있습니다. 희망을 가지세요 설치 CD가 올바르게 부팅된다는 말은 여러분이 만든 커널 또한 그렇게 동작하리란 것을 의미합니다. 설치 CD 프롬프트에서 다음 명령을 연속적으로 입력하면 설치된 젠투 리눅스의 영역으로 진입 가능합니다. 주의 깊게 잘못된 것이 무엇인지 확인해서 수정하세요. 손쉽게 고쳐지기를 바랍니다.

```
# mount /dev/sda5 /mnt/gentoo
# mount /dev/sda1 /mnt/gentoo/boot
# mount /dev/sda6 /mnt/gentoo/home
# swapon /dev/sda2
# mount -t proc none /mnt/gentoo/proc
# mount --rbind /sys /mnt/gentoo/sys
# mount --rbind /dev /mnt/gentoo/dev
# chroot /mnt/gentoo /bin/bash
# source /etc/profile
# export PS1="(chroot) $PS1"
```

시스템 설정

재부팅이 성공적이고, root로 로그인해서 쉘까지 진입을 했다면, 정말 축하드립니다! 여러분만의 젠투 시스템이 완성되었습니다. 젠투는 이제 자신의 커널을 이용해 여러분과 상호작용하며, 자가 운영, 자가 발전이 가능한 온전한 상태에 들어온 것입니다.

'온전한 상태'라는 듣기 좋은 완곡한 표현을 쓰긴 했지만, 뭐 사실은 아직은 황무지와 다름없습니다. X-윈도우도 설치하고 웹브라우저로 웹서핑도 하고 동영상도 시청하고 음악도 감상해야 합니다. 아직은 갈 길이 멀긴 합니다. 그렇지만 조금만 참으세요. 차차 젠투 리눅스로 할 수 있는 것이 많아집니다.

사족: 젠투 리눅스에 익숙해지기

저 같은 경우는 젠투에 많이 익숙해서 젠투가 아니면 그것이 오히려 불편합니다. 하지만 당장 윈도우를 쓰다가 리눅스로 온 이에게 리눅스는 많이 불편합니다. 당연히 불편할 수 밖에 없습니다. 게다가 젠투 리눅스와 같이 사용자가 모든 것을 선택하고 결정해야 하는 것이라면 갑자기 맞이하게 된 너무나 큰 자유도의 크기에 진절머리가 날 수도 있을 것이라 생각합니다.

그럴 리는 없겠지만, 결코 젠투 리눅스가(또는 모든 리눅스가) 뒤떨어지거나 못난 운영체제라고 생각하지는 말아주세요. 특히 윈도우에 익숙해신 분들은 아마도 조금만 불편해도 윈도우 생각이 간절해지실 것이고, 리눅스의 불편함에 분통을 터뜨리며 리눅스를 사용해보려 마음먹은 자신을 질책할 지도 모릅니다. 그러나 여러분의 선택은

그리 잘못된 것이 아닙니다.

저는 이런 상황을 이렇게 받아들이고 있습니다. 문제는 바로 **너무 빨리 문제를 해결하려고 조급해하는 것**입니다. 여러분이 취미로, 혹은 공부를 위해 이 문서를 읽고 젠투 리눅스를 설치한 것이라면 특히 더 강조하고 싶은 문제입니다. 대개의 문제는 리눅스가 그 자체에 있다기 보다는, 너무나 빨리 문제라고 인식하고, 그것을 너무 빨리 해결하려고 조급하게 행동하는 것이 문제입니다. 리눅스가 비난받을 이유는 거의 없다고 봅니다.

좀더 애착을 빨리 들일 수 있는 방법을 추천드립니다. 중요한 데이터를 리눅스가 관리하고 사용할 수 있도록 해 보세요. 자신의 개인 문서, 일기, 영화, 음악, .. 그 어떤 형태라도 좋습니다. 기존에 이용하던 OS 대신 리눅스에서 그러한 데이터를 사용할 수 있도록 환경을 조성하는 것이 좋습니다. 물론 백업은 필수입니다! 이렇게 하면 보다 빠르게 리눅스 사용에 익숙해질 것입니다.

원래 새로운 것을 받아들일 때는, 필연적으로 불편함을 느낍니다. 그것을 넘지 못하면 새로운 것에 익숙해질 수 없습니다. 넘어가실 때 넘어가시더라도 그 불편함이 잠깐의 불편함인지, 아니면 정말 리눅스가 사용 못할 정도로 실망스러운 운영체제였기 때문인지 조금 생각해 보아 주시기를 바랍니다. 그리고 ... 젠투 리눅스를 이용해 여기까지 왔습니다! 지금까지 컴파일한 패키지와 시간이 아깝지 않으세요? 😎 한 번 진득하게 마음을 먹고 익숙해보는 것도 나쁘지 않습니다. 그리고 그렇게 사용하다 보면 저절로 애착이 생기기 마련입니다.

물론 굳이 불편함을 감수해가면서 리눅스를 써야만 할 대의명분이 없다면 굳이 리눅스를 고집하지 않으셔도 됩니다. 리눅스 사용이 극기훈련도 아니니 불편함을 참지 못하시겠다면 원래 사용하던 편한 운영체제로 복귀하세요. 그 또한 여러분이 누릴 수 있는 자유입니다.

네트워크 설정하기

설치 때에는 임시적으로 네트워크를 설정했습니다. 그러나 이제는 부팅 때마다 자동으로 네트워크는 설정되어야 합니다.

유선 네트워크

고정 IP 주소를 사용할 경우 `/etc/conf.d/net` 파일을 열어 다음과 같이 설정합니다.

```
config_eth0="192.168.0.2 netmask 255.255.255.0"
routes_eth0="default via 192.168.0.1"
dns_servers_eth0="8.8.8.8 8.8.4.4"
```

유동 IP 주소를 사용할 것이라면 보다 간단합니다. `/etc/conf.d/net`을 열어

```
config_eth0="dhcp"
```

그리고 다음과 같이 작업해서 eth0가 동작하도록 설정합니다.

```
# cd /etc/init.d
# ln -s net.lo net.eth0
```

부팅 때마다 자동으로 주소를 받아오려면

```
# rc-update add net.eth0 boot
```

무선 네트워크

설치 때와 마찬가지로 wpa_supplicant 가 필요합니다. 설치 CD에는 있었지만 현재 우리 시스템에는 설치되어 있지 않을 것입니다.

```
# emerge net-wireless/wpa_supplicant
```

/etc/conf.d/net을 열어,

```
modules="wpa_supplicant"
wpa_supplicant_wlan0="-Dmadwifi"
```

/etc/wpa_supplicant/wpa_supplicant.conf 파일은 설치 때와 마찬가지로 설정합니다. 그리고 유선과 마찬가지로 ln -s 명령으로 링크를 만들고 rc-update 명령으로 부팅 때마다 자동으로 실행하도록 설정하면 됩니다.

호스트 네임 지정

다른 네트워크에 젠투 시스템이 설치된 PC의 이름을 알리는 데 필요합니다.

```
# nano -w /etc/conf.d/hostname
hostname='tux'
```

재부팅하면 쉘의 문자가 다름처럼 변경되었을 것입니다. 'tux' 부분은 여러분이 원하는 대로 변경하시면 됩니다.

```
tux ~ #
```

로케일 설정

로케일 설정을 합니다. 시스템이 사용자와 어떤 문자로 소통할지를 결정하는 것인데, 쉘에서는 한글을 쓰지 마세요. 그냥 영어를 사용하는 것이 좋습니다. 한글은 콘솔의 폰트가 없어 처참하게 망가지거든요.

```
# nano -w /etc/locale.gen
en_US.UTF-8 UTF-8
ko_KR.UTF-8 UTF-8
(저장)

# locale-gen
```

이렇게 하면 로케일이 생성됩니다. 그리고,

```
# nano -w /etc/env.d/02locale
LANG="en_US.UTF-8" # 바로 이 부분입니다. 이 부분을  ko_KR.UTF-8로 하진 마세요.
LC_COLLATE="C"
```

변경한 사항을 현재 시스템에 바로 반영하기 위해서는 다음 명령을 입력하면 됩니다.

```
# env-update
# source /etc/profile
```

리눅스는 그냥 영어로 쓰시는 것이 좋습니다. 에러가 날 경우 거의 구글을 통해 외국의 사이트들을 뒤지면서 문제를 해결해야 하는데, 에러 메시지가 한글로 나오면 오히려 찾기가 어렵습니다. 그리고 콘솔 환경은 반드시 영문으로 쓰세요. X윈도우의 터미널 에뮬레이터가 아닌 tty1~tty6에서는 한글이 와장창 깨져 나옵니다. 조만간 X 윈도우를 도입해 훨씬 편한 환경에서 작업할 예정인데 굳이 tty1~tt6에서 한글이 나오도록 애쓸 필요가 없습니다.

시스템 툴 설치

리눅스 시스템이면 거의 기본적으로 가질 도구들을 아직 설치하지 않은 상태입니다. 이번 장에서 차근차근 설치해 나가도록 합니다. 젠투의 공식 문서에는 설치 CD를 빼기 전에 이 과정을 모두 거치고 지나갑니다만, 강제 사항은 아니기에 이렇게 시스템을 완성한 후 차근차근 완성해 나가도 지장이 없습니다.

시스템 로거

시스템의 로거를 설치합니다.

```
# emerge syslog-ng logrotate
# rc-update add syslog-ng default
```

크론 데몬

정해진 시각에 정해진 작업을 수행하는 크론 데몬을 설치합니다.

```
# emerge vixie-cron
# rc-update add vixie-cron default
```

파일 인덱싱

시스템 파일을 인덱싱하려면, 다음 명령으로 설치합니다.

```
# emerge mlocate
```

SSH 기본설정

SSH 데몬을 부팅마다 실행시켜 원격으로 접속할 수 있도록 할 수 있습니다.

```
# rc-update add sshd default
# /etc/init.d/sshd start # 바로 지금 데몬을 활성화 하려면 이 명령을 입력합니다.
```

기본 사용자 생성

매번 root를 사용하는 것은 그다지 권장할 만한 일이 아닙니다. 일반 사용자를 만들어 보도록 하지요.

```
# useradd -m -G audio,wheel,users,video -s /bin/bash <new_user>
# passwd <new_user>
... # 패스워드를 입력
```

root에서 로그아웃하고 새로 만든 계정으로 로그인해 봅니다. 만일 루트로 작업해야 한다면,

```
$ su -
```

을 입력하고 루트의 패스워드를 입력하면 언제든지 루트로 변경될 수 있습니다. 다시 일반 사용자로 돌아가려면 다음과 같이 하면 됩니다.

```
# exit
```

sudo

우분투 리눅스의 경우 'sudo'를 기본적으로 사용하고, root의 로그인을 아예 막고 있습니다. 젠투 리눅스도 우분투처럼 sudo를 사용하고 아예 루트의 로그인조차 막아버리게 만들 수도 있지만, 데스크탑 PC에서 그렇게까지 깐깐할 필요는 없는 것 같습니다. 그냥 sudo 정도를 사용하고 사용자가 자신의 비밀번호를 입력하면 일시적으로 루트 권한을 가지도록만 조치하지요.

```
# emerge sudo  # 'sudo' 패키지를 설치
# visudo       # sudo 설정. 반드시 visudo를 사용해야 합니다.
```

아래와 같은 부분을 찾아 주석을 해제합니다.

```
## Uncomment to allow members of group wheel to execute any command
%wheel ALL=(ALL) ALL
```

새로운 일반 사용자 계정이 'wheel' 그룹에 속해 있다면 sudo 명령으로 일시적인 관리자 권한을 얻을 수 있습니다. 일반 사용자가 wheel group인지 확인하려면

```
# groups <user>
....
```

만일 유저가 wheel 그룹에 속해 있지 않다면 다음과 같은 명령으로 추가시키면 됩니다.

```
# gpasswd -a <user> wheel
```

만일 그래도 root의 로그인을 기어이 막고 싶으시다면, 이렇게 하시면 됩니다. 일반 사용자로,

```
$ sudo -s
password: *********
#           # 일반 사용자에서  root로 권한상승했습니다.
# passwd -l root      # root의 계정을 잠급니다. 여기까지 실행하면  root로 로그인하지 못합니다.
# passwd -u root      # root의 계정을 잠금 해제합니다.
# exit # 권한하강합니다.
$           # 일반 사용자
```

마치며

이제 여러분의 하드디스크에 젠투 리눅스가 확실히 살아 숨쉬게 되었습니다. 전원을 켜서 부팅을 하면 먼저 GRUB이 실행됩니다. 물론 젠투 리눅스도 선택할 수 있지만, 별도로 원하는 리눅스 혹은 윈도우로도 부팅이 가능합니다. 젠투 리눅스를 선택하면 우리가 직접 만든 커널이 메모리에 적재됩니다. 이 커널은 직접 우리 손으로 설정하고, 우리의 CPU가 컴파일을 해서 만든 것이지만, 다른 배포판에서 일괄적으로 복사되던 그 커널과 기능적으로 동일합니다.

우리의 시스템에는 bash 쉘이 기본적으로 설치되어 있습니다. 비록 검은 바탕의 투박한 콘솔이지만, 나름대로는 매우 짜임새 있고 티테일하게 잘 설정되어 있습니다. 관리자로도 접속되지만, 일상적으로 사용할 일반 계정도 마련해 보았습니다.

Portage와 ebuild는 강력한 시스템 관리 툴입니다. 네트워크에 접속되어 있다면 'emerge' (Ebuild MERGE) 명령어를 통해 젠투의 소프트웨어 저장소로부터 원하는 프로그램을 다운로드 받을 수 있습니다. Ebuild는 매우 유연하고도 편리하게 프로그램을 소스로부터 빌드해냅니다. 프로그램 간의 의존성도 자동으로 파악해 주므로 여러분들이 소프트웨어를 찾아 이곳저곳 헤매지 않아도 됩니다.

가장 인상적인 점이라면, 이 모든 것이 다 여러분들이 직접 선택해 만들어낸 결과물이란 점입니다. 대단하지 않습니까? 물론 정말 모든 것을 온전히 우리의 손으로 만든 것은 아니고[7] 실제로는 설치 단계에서 곳곳에서 여러 가지 젠투의 편리한 도구를 사용해 설치를 진행하긴 했습니다. 그러나 설치 거의 모든 단계단계에서 우리 스스로가 직접 디테일한 사항을 고려하고 결정하였습니다. 이전 make.conf에서 언급한 '밖으로 비치는 주방을 갖춘 식당'은 바로 이를 두고 한 말입니다.

아 하나 잊은 것이 있습니다. 디스크에 stage3와 portage snapshot 파일이 남아 있다면 삭제하세요. 그리고 이제 젠투 시스템은 막 시작했을 뿐입니다. X-윈도우 시스템도 붙이고 여러 어플리케이션도 설치해야 하고, 이런저런 커스터마이징도 해서 때깔나는(!) 모습으로 만들어내기도 해야 합니다. 문서의 분량이 상당히 많아지므로 이것은 별도의 문서로 꾸미도록 하겠습니다. 읽어주셔서 감사합니다.

[1] `less /usr/portage/profiles/use.desc`로 열람할 수 있습니다
[2] 리브레오피스나 파이어폭스 같은 컴파일에 지나친 시간이 걸리는 몇몇 패키지는 마치 데비안의 apt처럼 미리 컴파일된 바이너리 패키지를 받아 설치하도록 하는 옵션입니다
[3] 각 파일의 내용에는 문제가 없습니다
[4] 7zip이나 rar은 인코딩을 고려하여 압축 생성/해제를 하므로 큰 문제가 없습니다
[5] http://www.gentoo.org/doc/en/gentoo-amd64-faq.xml#masked
[6] nvidia는 단지 설명을 위해 예를 든 것 뿐입니다. 현재는 X윈도우를 설치할 단계가 아닙니다. 실제로 nVidia의

그래픽 카드를 사용해고 있어도 현재 시스템에는 아직 설치되지 않았을 것입니다.
[7] 정말 모든 것을 하나하나 다 직접 빌드하는 특이한 리눅스가 있습니다. LFS (Linux From Scratch), 이것은 정말로 끈기와 인내를 가지고 해야 합니다.