The Linux Bootdisk HOWTO

FawcettTom

Bootdisk-HOWTO@linuxdoc.org

도운이

김현종

polsci@hitel.net

v4.4, June 2001

본 문서는 리눅스용 부트/루트디스켓들을 제작하는 방법을 설명합니다. 이렇게 제작한 디스크는 복구용 디스크로 사용하거나 새로운 시스템 구성을 시험해 볼 때 쓸 수 있습니다. 부트디스켓을 직접 제작하려면 시스템 관리 일반에 대해 어느정도 지식이 있어야 합니다. 단순히 응급상황에 대비한 복구디스크가 필요한 것이라면 부록의 Appendix A.1 부분을 보십시오.

1. 머리말

> 중요**:** 본 문서는 이미 한물간 것이 되었을지도 모르니 문서 첫부분의 날짜가 6개월 이전이라면 Bootdisk-HOWTO 홈페이지 (http://www.croftj.net/~fawcett/Bootdisk-HOWTO/index.html)에서 새 버전의 문서를 확인해 보시기 바랍니다.

본 문서는 txt 포맷으로 보셔도 됩니다만 문서 내에 몇가지 기호를 사용했으므로 포스트스크립트 포맷이나 HTML 포맷, 혹은 PDF 포맷이 더욱 보기 편할 것입니다.

1.1. 본 문서의 버전에 대해

Graham Chapman 씨가 최초의 Bootdisk-HOWTO 문서를 쓰셨고 3.1 버전까지 담당하셨습니다. Tom Fawcett 씨가 커널 2.0 때부터 공동저자로 참가하셨고 현재 본 문서를 관리하고 있습

니다. Chapman 씨는 리눅스 공동체에서 더이상 활동하지 않고 있으며 현재 그의 근황은 알려지지 않고 있습니다.

본 문서는 인텔 프로세서 기반의 리눅스를 대상으로 합니다. 다른 프로세서용의 리눅스에도 이 글의 많은 부분이 적용되겠지만 필자는 이에 대해서는 직접적인 경험도 없고 잘 알지도 못합니다. 다른 플랫폼상의 부트디스크에 경험이 많으신 분은 필자에게 연락을 좀 주십시오.

1.2. 아직 부족한 부분들

1. `libc.so`, busybox, uClib 등의 덩치큰 프로그램들을 부트디스크에 포함시키는 방법
2. 부트디스크를 테스트하는 방법으로서의 사용자모드-리눅스에 대한 내용. 필자도 아직 연구중임.
3. 배포본들의 부트디스크를 다시 분석해서 "전문가들이 사용하는 방법" 부분을 갱신
4. 배포본의 부트디스크를 업그레이드하는 방법 부분을 삭제. 이 부분은 사용자를 더 곤경에 빠트리기 쉽다.

1.3. 사용자의 의견과 도와주신 분들

좋은 평이든 아니든 이 문서에 대한 여러분의 의견을 환영합니다. 글의 내용이 정확하고 믿을만한 것이 될 수 있도록 저는 최선을 다했습니다. 만일 틀린 부분이나 소홀한 부분을 발견하신다면 연락해 주십시오. 연락주실 때는 읽으신 해당 문서의 버전을 같이 알려주십시요.

좋은 제안과 정정을 해주신 많은 분들께 감사드립니다. 그분들의 도움으로 인해, 저희 혼자서 하는 것보다 훨씬 좋은 내용이 될 수 있었습니다.

의견이나 정정할 부분이 있다면 위에 적힌 필자의 e-mail 주소로 보내주십시요. 부디 질문을 보내시기 전에 먼저 **7**절 부분을 읽어보시기 바랍니다**.** 필자에게 디스크 이미지를 보내지는 말아주십시요**.**

1.4. 본 문서의 배포에 대해

2. 소개

리눅스 부트 디스크가 유용한 상황은 많습니다. 새로운 커널을 테스트하는 경우, 디스크의 문제를 복구할 경우(부트섹터를 날렸거나 디스크 헤드가 망가진 경우 등등), 망가진 시스템을 고칠 경우, 핵심적인 시스템 파일((`libc.so` 따위)들을 안전하게 업그레이드하는 경우등에 유용하게 사용할 수 있습니다.

부트 디스크는 다음 방법을 통해 구할수 있습니다.

- 슬랙웨어 등의 배포본에 있는 부트디스크를 이용한다. 이런 것을 쓰면 적어도 부팅만큼은 확실하게 할 수 있다.
- 복구용도로 설계된 복구패키지의 디스크를 이용한다.
- 각각의 디스크들이 어떤 작업을 수행하는지를 이해한 후 직접 부트 디스크를 제작한다.

위의 마지막 방법을 택하면, 고장이 난다해도 그 원인을 알고 있으므로 직접 수리해낼 수 있습니다. 게다가 리눅스 시스템이 어떻게 작동하는가를 이해하는데 더할나위 없이 좋은 방법입니다.

이 문서는 여러분이 리눅스 시스템 관리상의 몇 가지 개념들에 이미 익숙해있다고 가정합니다. 예를 들면 디렉토리, 파일시스템, 플로피 디스켓에 대해 알고 있어야만 합니다. **mount** 명령과 **df** 명령의 사용법도 알고 있어야 합니다. `/etc/passwd` 와 `fstab` 파일이 왜 필요하고 어떤 형태인지 알고 있어야 합니다. 이 HOWTO 문서에 등장하는 대부분의 명령들은 루트 권한으로 실행해야 한다는 점에 주의하십시요.

간단한 구상만을 바탕으로 바로 부트디스크 제작에 착수한다면 일이 꽤 어려워질 수 있습니다. 만일 리눅스 FAQ, 리눅스 설치 HOWTO, 리눅스 설치가이드 등등의 관련문서들을 읽어본적이 없다면 직접 부트 디스켓을 제작하는 것은 무리입니다. 응급상황에 대비해 확실히 동작하는 복구용 부트디스크가 필요할 뿐이라면 이미 만들어져 있는 것을 다운 받는 쪽이 훨씬 편합니다. 부록 Appendix A.1 부분에 어디에서 얻을 수 있는지 적혀있습니다.

3. 부트디스크와 부팅 과정

부트디스크란 기본적으로 플로피 디스켓 한장에 쏙 들어가는 축소판 리눅스 시스템입니다. 부트디스크는 완전한 풀 사이즈의 리눅스 시스템의 기능 중 많은 부분을 그대로 수행할 수 있

어야 합니다. 부트 디스크 제작에 앞서, 당신은 리눅스의 부팅 과정의 기본 원리를 이해해야 만 합니다. 여기서는 이 문서의 내용을 이해할수 있을 정도의 기본적인 내용만을 설명할 것입니다. 많은 세부적인 사항과 기타 옵션들에 관한 것은 생략했습니다.

3.1. 부팅 과정

모든 PC 시스템들은 롬(정확히는 **BIOS**)내의 코드를 실행시키는 것으로 부팅을 시작합니다. 이 코드는 부트 드라이브의 섹터 0, 실린더 0 부분을 읽어들입니다. 부트 드라이브는 보통 첫 번째 드라이브(도스로 말하자면 `A:`, 리눅스로 말하자면 `/dev/fd0`)를 말합니다. 그 다음, BIOS 는 읽어들인 이 섹터의 내용을 실행합니다. 대부분의 부트 가능한 디스크들은 섹터 0, 실린더 0 영역에 다음 내용 중 한 가지를 담고 있습니다.

- LILO 등과 같은 부트로더(boot loader)의 코드. 부트로더는 커널을 찾아 메모리에 로드한 후 실행시키는 방식으로 부트를 시작합니다. 아니면,
- 리눅스 등과 같은 운영체제 커널의 시작 부분.

만일 리눅스 커널이 디스켓에 직접 복사된 경우(raw copy)라면 디스크의 첫번째 섹터는 리눅스 커널 그 자체의 첫번째 섹터가 됩니다. 이 첫번째 섹터는 부트 디바이스로부터 커널의 나머지 부분을 계속 읽어들임으로써 부트 프로세스를 진행합니다.

일단 커널이 완전히 로드되면, 커널은 기본적인 디바이스들과 그 내부 데이터 구조를 초기화시킵니다. 초기화가 완료되면 커널 이미지내의 특정한 위치에 있는 램디스크 워드라는 것을 읽습니다. 이 워드는 루트 파일시스템을 어디에서 어떻게 찾아야 하는지를 나타내고 있습니다. 루트 파일시스템이란 단순히 "/ 에 마운트되는 파일시스템을 말합니다. 커널은 어디에서 루트 파일시스템을 찾아야 하는지를 알아야만 합니다. 만일 커널이 그 위치에서 로드 가능한 이미지를 찾지 못한다면 시스템은 멈춰버리게 됩니다.

어떤 부팅의 경우에는 -- 주로 디스켓에서 부팅하는 경우 -- 루트 파일시스템을 램디스크로 로드하기도 합니다. 램디스크란 시스템의 램의 일부를 마치 디스크처럼 취급하는 것입니다. 램은 플로피디스크보다 수천배 이상 빠르기 때문에 시스템을 빠르게 구동시킬 수 있습니다. 또한, 루트 파일시스템을 압축시켜 플로피에 담은 경우, 커널은 플로피로부터 이 압축을 풀면서 램디스크로 로드할 수 있습니다. 따라서 좀 더 많은 파일들을 디스켓 상에 압축시켜 둘 수 있습니다.

일단 루트 파일시스템이 로드되어 마운트되면 다음과 같은 메시지를 볼 수 있을 것입니다.

```
VFS : Mounted root (ext2 filesystem) readonly.
```

일단 시스템이 루트 파일시스템을 로드하는데 성공하면, 다음으로 루트 파일시스템에 있는 `init` 프로그램을 찾아 실행을 시도합니다(`/bin` 이나 `/sbin` 에 들어있습니다). `init` 는 그 설정파일인 `/etc/inittab` 에서 `sysinit` 라인을 찾아 그에 해당하는 이름의 스크립트를 실행시킵니다. `sysinit` 스크립트는 보통 `/etc/rc` 나 `/etc/init.d/boot` 같은 것들입니다. 이 스크립트는 쉘 명령어로 짜여진 것으로서 하드디스크에 대해 **fsck** 를 실행하거나, 필요한 커널 모듈들을 로드하기도 하고, 스와핑을 초기화시키고, 네트웍을 초기화시키며 `/etc/fstab` 에 적힌 디스크들을 마운트하기도 합니다.

이 스크립트는 대개 다른 여러가지 스크립트들을 또 동작시킵니다. 즉, 초기화 과정을 모듈화시킨 것입니다. 예를 들면, 일반적인 SysVinit 구조에서는 `/etc/rc.d/` 디렉토리 밑에 복잡한 구조의 하위디렉토리가 있고 각각의 하위디렉토리에는 수많은 시스템 서비스들을 어떻게 온오프 시키는지를 정해놓은 파일들이 있습니다. 하지만 부트디스크에서 사용하는 sysinit 스크립트는 보통 매우 간단한 것입니다.

sysinit 스크립트의 실행이 끝나면 다시 **init** 프로세스로 조종권이 돌아오고, 이번에는 **default runlevel** 단계로 들어갑니다. default runlevel 은 `inittab` 파일내에 `initdefault` 키워드로 지정되어 있습니다. runlevel 라인은 주로 콘솔이나 tty 를 통한 통신을 책임지는 **getty** 같은 프로그램을 지정합니다. 우리에게 익숙한 "`login:`" 프롬프트 따위를 출력해 주는 것이 바로 **getty** 프로그램입니다. 이제 **getty** 프로그램은 로그인 인증 처리와 user 세션을 마련해주는 **login** 프로그램을 구동시킵니다.

3.2. 디스크의 종류

기본적인 부팅 과정을 살펴보았으므로 이제 필요한 디스크들을 종류별로 정의해봅시다. 디스크를 4 가지 종류로 나눠봅시다. 이 문서에서 "디스크" 라는 단어는 특별한 언급이 없는 한 플로피디스켓을 의미합니다만 대부분의 경우 그 내용은 하드디스크에도 그대로 적용될 수 있습니다.

boot

부트 가능한 커널을 포함한 디스크. 이 디스크는 커널을 부트시키는 용도로 사용되며, 이렇게 로드된 커널은 또다른 디스크에 위치하고 있는 루트 파일시스템을 로드할 수 있습니다. 보통, 부트디스크상의 커널은 루트 파일시스템이 어디에 위치하고 있는지를 미리 지정받아 알고 있어야만 합니다.

대부분의 경우 부트디스크는 다른 디스켓상의 루트 파일시스템을 로드하게 되지만, 때로는 하드디스크에 있는 루트 파일시스템을 로드하도록 설정할 수도 있습니다. 이런 기법은 주로 새로운 커널을 테스트해볼 때 사용됩니다(사실 "**make zdisk**" 명령은 커널 소스코드로부터 자동으로 이런 류의 부트디스크를 만들어줍니다).

root

리눅스 시스템 구동에 필요한 파일들을 가진 파일시스템을 루트 파일 시스템이라 하며, 이 루트 파일시스템을 담은 디스크가 루트 디스크입니다. 루트 디스크가 꼭 커널이나 부트로더를 함께 담고 있을 필요는 없습니다.

일단 커널이 부트된 상태라면 루트 디스크는 다른 어떤 디스크도 필요없이 독자적으로 시스템을 운용할 수 있습니다. 보통, 루트 디스크는 자동적으로 램디스크로 복사됩니다. 램디스크를 사용하면 루트 디스크에 대한 액세스가 훨씬 빠르며, 또한 디스크 드라이브를 비울수 있어 거기에 유틸리티 디스크를 넣을 수 있습니다.

boot/root

커널과 루트 파일시스템을 한장에 모두 담고있는 디스크를 말합니다. 바꾸어 말하면, 하

드디스크 없이도 이 디스크는 리눅스 시스템을 부트하고 운용하는데 필요한 모든 것을 다 가지고 있는 것입니다. 이러한 타입의 디스크의 장점은 콤팩트하다는 것입니다 –– 필요한 모든 것이 한 장의 디스크에 들어갑니다. 하지만 리눅스의 모든 것이 점차 커져가는 추세에 있기 때문에, 비록 압축해서 담을 수 있다고는 해도 한 장의 디스켓에 모든 것을 담는 것은 점점 어려워지고 있습니다.

utility

그 밖의 여러가지 데이터를 담은 디스켓으로서, 파일시스템을 담고는 있지만 루트 파일시스템으로서 마운트되는 것은 아닙니다. 이것은 추가적인 데이터 디스크입니다. 루트 디스크 한장에 다 담기 힘들 경우, 여분의 유틸리티들을 이 디스크에 담게 됩니다.

일반적으로 "부트디스크를 제작"한다고 말할 때는 boot(커널) 와 root(파일들) 부분을 모두 만드는 것을 뜻합니다. 두 부분을 하나의 디스크에 담을 수도 있고(boot/root 디스크) 두 장의 디스크로 분리하여 담을수도 있습니다(boot + root 디스크들). 아마도 boot 디스켓과 root 디스켓을 각각 따로 만들고 그래도 모자라는 경우 한두 장의 utility 디스켓을 더 만드는 것이 복구 디스켓들을 제작하는 가장 유연한 방법일 것입니다.

4. 루트 파일시스템의 제작

루트 파일시스템을 만들 때는 시스템 구동에 필수적인 파일들을 고르는 작업이 필요합니다. 이 절에서는 압축된 루트 파일시스템의 제작법을 설명합니다. 별로 많이 쓰이지는 않지만 압축안된 파일시스템을 디스켓상에 만들어 직접 루트로 마운트시키는 방법도 가능합니다. 이 방법은 8.2절 부분에서 설명합니다.

4.1. 개요

루트 파일시스템은 풀 사이즈의 완전한 리눅스 시스템을 지원하기 위한 모든 것을 갖추어야 합니다. 이를 위해서는 리눅스 시스템의 최소요건만큼은 루트디스크에 반드시 구비되어야 합니다.

- 기본적인 파일 시스템 구조
- 최소한도의 디렉토리들 : `/dev`, `/proc`, `/bin`, `/etc`, `/lib`, `/usr`, `/tmp`,
- 기본적인 유틸리티들 : `sh`, `ls`, `cp`, `mv`, 기타 등등
- 필수적인 설정 파일들: `rc, inittab, fstab`, 기타 등등
- 디바이스 : `/dev/hd*, /dev/tty*, /dev/fd0`, 기타 등등
- 유틸리티 프로그램에 필요한 기본적인 함수들을 제공하는 런타임 라이브러리.

물론, 어떤 시스템이 됐든간에 원하는 프로그램을 실행시킬 수 있을 때 비로소 이용가치가 있는 거겠지요. 그런 점에 미루어 볼때, 루트 디스켓으로 다음과 같은 작업을 할수 있어야 할 것입니다.

- 다른 드라이브에 있는 파일 시스템을 체크하는 작업. 예를 들어 하드디스크에 담긴 루트 파일시스템을 점검하고자 한다면, 루트디스켓 시스템을 마운트하는 등의 방법을 써서, 점검하고자 하는 하드디스크가 아닌 다른 디스크에서 리눅스를 부팅시킬 필요가 있습니다. 이렇게 하면 하드디스크가 마운트되지 않은 상태에 있게 되므로 이제 **fsck** 명령으로 하드디스크를 점검할 수 있습니다.
- `cpio`, `tar`, `gzip`, `ftape` 등의 archive 및 압축 유틸리티를 써서 백업으로부터 원래의 루트드라이브의 전부 혹은 일부를 복구하는 작업.

이제 압축 파일시스템을 어떻게 만드는지 설명하겠습니다. 압축 파일시스템이라는 말은 파일시스템이 디스크에 압축된 상태로 있다가 부트시에 램디스크로 압축이 풀리면서 복사되는 것을 말합니다. 압축 파일시스템을 쓰면 표준 1440K 디스켓상에 훨씬 많은 파일(약 6메가 가량)들을 넣을 수 있습니다. 파일시스템이 디스켓의 용량보다 훨씬 크기 때문에, 디스켓 위에 파일 시스템을 직접 작성하는 것은 불가능합니다. 일단 다른 곳에서 파일 시스템을 완전히 만든 후, 이를 압축한 다음, 그 압축된 것을 디스켓에 복사해넣어야 합니다.

4.2. 파일 시스템 만들기

압축된 루트 파일시스템을 만들기 위해서는 압축전에 먼저, 필요한 모든 파일들을 담을 수 있는 충분한 크기의 빈 공간이 필요합니다. 약 4 메가바이트 가량을 담을수 있는 디바이스가 필요합니다. 몇 가지 방법이 있습니다.

- 램디스크를 사용하는 방법입니다(DEVICE = `/dev/ram0`). 이 경우 메모리 일부가 가상의 디스크 드라이브로 설정됩니다. 램디스크는 필요한 크기의 파일시스템을 담을 수 있을 크기는 되어야 합니다. 만일 LILO 를 쓰고있다면 설정파일(`/etc/lilo.conf`)에 `RAMDISK = nnn` 같은 라인이 있는지 확인해 보세요. 이 라인은 하나의 램디스크가 가질수 있는 최대 램 크기를 정하는 것입니다. 디폴트는 4096K 인데 이 정도면 충분할 것입니다. 만약 시스템의 램이 8 MB 미만이라면 램디스크를 사용하는 방법은 피하는 것이 좋습니다.

  `/dev/ram0` 이나 `/dev/ram`, 혹은 `/dev/ramdisk` 등의 디바이스를 가지고 있는지 확인하십시오. 만일 없다면 **mknod** 명령(major number 1, minor 0)으로 `/dev/ram0`를 만들어 주어야 합니다.
- 만약 쓰지 않는 수 메가바이트 정도의 하드디스크 파티션이 있다면 이를 이용해도 됩니다.
- 루프백 디바이스를 이용할 수도 있습니다. 루프백 디바이스는 파일 하나를 마치 디바이스처럼 취급할 수 있게 해줍니다. 즉, 파일 한개를 마치 하나의 하드디스크 파티션처럼 인식시키는 것입니다. 루프백 디바이스를 이용해서 하드디스크 상에 3 메가바이트 가량의 파일을 만든 후 이 위에 파일 시스템을 만들 수 있습니다.

man losetup 이라고 타이핑해보십시요. 루프백 디바이스의 사용법이 출력될 것입니다. 만일 **losetup** 이 없다면 `ftp://ftp.win.tue.nl/pub/linux/utils/util-linux/` 디렉토리에서 `losetup` 과, 이에 맞는 버전의 **mount**, **umount** 바이너리가 들어있는 `util-linux` 패키지를 받아 설치하면 됩니다.

시스템에 루프 디바이스(`/dev/loop0`, `/dev/loop1` 등등)가 없다면 "**mknod /dev/loop0 b 7 0**" 명령으로 만들어야만 합니다. 적절한 **mount** 및 **umount** 바이너리들을 설치했다면, 아래의 명령을 써서 하드디스크 상에 충분한 크기의 임시파일을 만드십시오(예를 들면 `/tmp/fsfile`).

```
dd if=/dev/zero of=/tmp/fsfile bs=1k count=nnn
```

위의 명령으로 `nnn`-block 의 파일을 만들 수 있습니다.

앞으로 DEVICE 라는 부분이 나오면 대신 위의 파일이름을 써주세요. 또한, `-o loop` 옵션을 주어 mount 프로그램에게 를 마운트함을 알려야 합니다. 예를 들면 다음과 같습니다.

```
mount -o loop -t ext2 /tmp/fsfile /mnt
```

위의 명령은 루프백 디바이스를 통해 `/tmp/fsfile` 를 마운트포인트 `/mnt` 에 마운트 시킵니다. **df** 명령으로 마운트 되었는지 확인해 볼수 있습니다.

위에서 말한 세가지 방법 중 어느 하나를 선택하기로 마음먹었다면 이제 DEVICE 에 다음 명령을 주세요.

```
dd if=/dev/zero of=DEVICE bs=1k count=4096
```

이 명령은 디바이스의 내용을 모두 0 으로 채웁니다.

> 중요**:** 디바이스를 0 으로 채우는 작업이 중요한 이유는 디바이스상의 파일 시스템은 나중에 압축되게 되므로 사용되지않는 모든 영역은 0 으로 채워야 최대한으로 압축할수 있기 때문입니다. 파일시스템상에서 파일을 지우거나 이동시킬때는 이 점을 꼭 명심하십시요. 파일시스템은 해당 블록의 할당을 회수함으로서 화일을 삭제, 이동시킵니다만 이때 그 블록의 내용까지 다시 **0** 으로 채워주는 것은 아닙니다**.** 파일의 삭제및 복사가 빈번한 경우, 최종적인 당신의 압축 파일시스템은 훨씬 커져버릴 수 있습니다.

그 다음, 파일시스템을 만듭니다. 리눅스 커널을 자동으로 램디스크로 복사되도록 해주는 루트 디스크용 파일시스템은 minix 와 ext2 파일시스템 단 두가지 뿐입니다. 이중에서 ext2 파일시스템이 보다 선호되는 파일 시스템입니다. ext2 를 쓰면 `-N` 옵션을 주어 디폴트값보다 더 많은 inode 를 설정할 수 있어 편리합니다. `-N 2000` 정도로 설정하면 inode 가 부족해지는 일은 없을 것입니다. 그밖에, `/dev` 디렉토리 밑의 불필요한 파일들을 제거해서 inode 를 절약하는 방법도 있습니다. **mke2fs** 는 디폴트로 1.44 Mb 디스켓에 360 개의 inode를 생성합니다. 필자가 쓰는 복구용 루트디스켓에는 120개 의 inode 가 있고 이 정도로 충분하지만 만일 당신이 `/dev` 디렉토리내의 디바이스 파일들을 전부 포함시키려 한다면 필요한 inode 수는 360 개를 쉽게 초과해 버립니다. 압축 루트파일시스템을 사용하면 보다 큰 파일시스템을 담을 수 있고 따라서 디폴트로 보다 많은 inode를 쓸 수 있습니다만 그래도 역시 파일의 수를 줄이거나 inode 수를 늘일 필요가 있을 것입니다.

따라서 다음과 비슷한 명령이 필요합니다.

```
mke2fs -m 0 -N 2000 DEVICE
```

(루프백 디바이스를 사용한다면 위의 DEVICE 대신 파일이름을 써야 합니다.)

mke2fs 명령은 자동으로 사용가능한 용량을 인지하고 그에 맞춰 파일시스템을 설정합니다. "-m 0" 파라메터는 **mke2fs** 로 하여금 root 용 공간을 할당하지 못하게 함으로써 사용가능한 디스크 용량을 더 많이 확보합니다.

이제 디바이스를 마운트하세요.

```
mount -t ext2 DEVICE /mnt
```

(만약 /mnt 디렉토리가 없다면 사전에 마운트포인트가 될 /mnt 디렉토리를 만들어 주어야 만 합니다). 앞으로 우리가 만들 모든 디렉토리들은 /mnt 아래에 있는 것으로 가정하겠습니다.

4.3. 파일시스템의 구성

다음은 아마도 당신의 루트 파일시스템에 들어있어야할 최소한의 디렉토리들입니다. [1]

- /dev -- 디바이스들이 위치합니다. I/O 에 필요합니다
- /proc -- proc 파일 시스템에 필요한 디렉토리
- /etc -- 시스템 설정파일들이 위치합니다.
- /sbin -- 시스템에 없어서는 안될 필수 바이너리들이 위치합니다.
- /bin -- 시스템의 일부로 간주되는 기본적인 바이너리들이 위치합니다.
- /lib -- 런타임 지원의 공유라이브러리들이 위치합니다.
- /mnt -- 다른 디스크들을 관리하기 위한 마운트포인트
- /usr -- 그밖의 여러 유틸리티와 응용프로그램들이 위치합니다.

루트 화일시스템상에서 위의 디렉토리 중 3 개는 빈 디렉토리가 됩니다. 따라서 그 3 개는 **mkdir** 명령으로 디렉토리만 만들어 주면 됩니다. /proc 디렉토리는 단순히 proc 파일 시스템이 위치하게 되는 장소(stub)일 뿐입니다. /mnt 와 /usr 디렉토리들은 boot/root 시스템이 가동된 후에야 사용되는 마운트포인트입니다. 따라서 다시 말씀드리지만 이 3 개의 디렉토리는 단지 디렉토리만 만들어주면 됩니다.

이제 나머지 4 개의 디렉토리에 대해 설명드리겠습니다.

4.3.1. /dev

/dev 디렉토리에는 시스템이 사용하는 모든 디바이스들 각각에 대응하는 특수파일들이 위치하게 됩니다. /dev 디렉토리는 모든 리눅스 시스템에 반드시 있어야만 하는 강제사항입니다. /dev 디렉토리 자체는 보통의 디렉토리와 다를바 없으므로 **mkdir** 명령어로 그냥 만들어

주면 됩니다. 하지만 /dev 디렉토리 내에 위치하는 디바이스 파일들 만큼은 특수한 파일들이므로 **mknod** 명령을 사용하는 특수한 방식으로 만들어주어야 합니다.

하지만 보다 간단한 방법이 있습니다. — 당신 시스템의 하드디스크에 있는 /dev 디렉토리에서 필요한 디바이스 화일들을 복사해오는 것입니다. 이때 유념해야 할 것은 특수 디바이스 파일들을 복사해 올 때는 -R 옵션을 써서 복사해야 한다는 점입니다. 이렇게 해야 디렉토리가 복사될 때 파일들의 내용들은 복사되지 않게 됩니다. 대문자 **R** 임에 주의하십시오. 명령어의 예는 다음과 같습니다.

```
cp -dpR /dev/fd[01]* /mnt/dev
cp -dpR /dev/tty[0-6] /mnt/dev
```

위의 명령은 디스켓이 /mnt 에 마운트 되었다고 가정한 것입니다. dp 스위치들은 각각 심볼릭 링크가 복사될 때 타겟파일들이 복사되는 것이 아니라 링크로서 복사되도록 해주며 원래의 파일 속성들이 그대로 유지된 채 복사되도록 해줍니다. 따라서 파일 소유권 정보가 그대로 유지됩니다.

어려운 방법으로 해보고 싶다면 ls -l 로 원하는 디바이스의 메이저와 마이너 디바이스 넘버를 출력해서 확인한 후 mknod 명령을 써서 직접 그대로 만들어 주면 됩니다.

디바이스 화일들을 다 만들었다면, 필요한 특수 디바이스들이 복구디스켓에 제대로 들어갔는지 확인하십시요. 예를 들어 ftape 명령은 테이프 디바이스를 사용하므로, 당신이 부트 디스크를 써서 테이프 드라이브 장치들을 액세스할 작정이라면 테이프 장치에 관련된 디바이스 화일들을 다 포함시켜야 할겁니다.

각각의 특수 디바이스 파일은 하나씩의 inode 를 필요로 하기 때문에 경우에 따라서는 inode 가 부족할 수 있습니다. 특히나 디스켓 파일 시스템에서는 더욱 그렇습니다. 따라서 필요한 디바이스들만 골라서 포함시키십시요. 예를 들어 SCSI 디스크를 가지고 있지 않다면 /dev/sd* 로 시작하는 디바이스 파일들은 무시해도 좋습니다. 마찬가지로, 시리얼 포트를 사용할 일이 없다면 /dev/ttyS* 로 시작하는 디바이스 파일들은 무시해도 좋습니다.

만일, 루트 파일시스템을 만들다가 No space left on device 이라는 에러가 떴는데 막상 df 명령을 내려보면 사용가능한 공간이 아직 남아있는 경우라면 아마도 inode 를 다써버린 경우일 겁니다. df -i 명령은 inode 의 사용상태를 보여줍니다.

> 중요**:** /dev 디렉토리에 다음 화일들은 반드시 포함되어야 함을 명심하세요: console, kmem, mem, null, ram0, tty1.

4.3.2. /etc

/etc 디렉토리에는 중요한 설정파일들이 위치합니다. 대부분의 시스템에 있어 설정파일들은 다음 세가지 정도로 분류할 수 있습니다.

1. 어떤 경우든 반드시 필요한 파일들. 예를 들면 rc, fstab, passwd 등등.
2. 반드시는 아니지만 대체로 필요하다고 생각되는 파일들.

3. 그외 필요한 잡동사니들

필수적인 파일인지 아닌지는 대략 다음과 같이 확인해 보실 수 있습니다.

```
ls -ltru
```

이 명령은 마지막으로 액세스된 시간순서로 파일들을 출력합니다. 따라서 최근에 액세스된 적이 없는 파일들은 루트디스켓에서 제외시킬수 있을 것입니다.

필자의 루트디스켓에는 약 15 개 정도의 설정파일이 들어있습니다. 용도에 따라 세가지 정도로 나누어 보겠습니다.

1. boot/root 시스템을 설정하는데 꼭 필요한 설정파일들 :
   a. `rc.d/*` -- 시스템 기동 및 런레벨 변경 스크립트들
   b. `fstab` -- 마운트될 파일 시스템의 리스트
   c. `inittab` -- **init** 프로세스에 대한 파라메터들이 담겨있습니다. **init** 는 부트시의 첫번째 프로세스입니다.

2. boot/root 시스템의 정돈에 필요한 설정파일들 :
   a. `passwd` -- 사용자, 홈 디렉토리 등등이 기록된 극히 중요한 리스트
   b. `group` -- 사용자 그룹들
   c. `shadow` -- 사용자들의 패스워드. 사용하지 않을 수도 있습니다.
   d. `termcap` -- 터미널의 기능에 대한 데이터베이스

   보안이 중요한 경우라면 사용자 패스워드가 시스템을 떠나 존재하지 않도록 `passwd` 와 `shadow` 는 디스켓으로 복사해오지 않는 것이 좋습니다. 이렇게 해두면 디스켓으로 부팅시 원치않는 사용자의 로그인을 막을 수 있습니다.

   `passwd` 는 적어도 `root` 만큼은 포함하고 있어야 합니다. 만일 다른 사용자들도 이 디스켓으로 로그인할 수 있게 하려면 이 화일로 그 사용자의 홈 디렉토리와 쉘을 마련해 주어야 합니다.

   `termcap`, 즉 터미널 데이터베이스는 보통 수백 킬로바이트 가량 됩니다. boot/root 디스켓에는 당신이 주로 사용하는 터미널들의 엔트리만 남기고 나머지는 삭제하세요. 보통은 `linux` 혹은 `linux-console` 엔트리만 남기면 될겁니다.

3. 나머지 기타 설정파일들. 때로 필요한 경우가 있어서 필자는 남겨두고 있습니다.

필자는 이 중에서 두 가지 파일만큼은 반드시 설정해주는데 그 내용은 무척이나 간단합니다.

- `rc` 에는 다음 내용이 들어있습니다.

```
#!/bin/sh
/bin/mount -av
```

```
/bin/hostname Kangaroo
```

화일이 실행가능한 상태인지 확인하십시요. 화일 첫머리의 첫번째 라인이 "#!" 으로 시작해야 합니다. 화일의 절대경로가 맞는지도 확인하십시오. 사실 **hostname** 명령은 실행시키지 않아도 무방합니다 — 해주면 더 멋져 보일 뿐입니다.

- `fstab` 에는 최소한 다음 내용은 들어있어야 합니다.

```
/dev/ram0       /           ext2    defaults
/dev/fd0        /           ext2    defaults
/proc           /proc       proc    defaults
```

시스템에서 현재 사용하고 있는 `fstab` 의 엔트리를 베껴와도 좋습니다. 하지만 당신의 하드디스크 피티션을 자동으로 마운트하게 해서는 안됩니다. 하드디스크의 각 파티션에 `noauto` 키워드를 써 주세요. 하드디스크가 손상되었거나 죽어버려서 부트디스크를 사용해야할 경우도 있기 때문입니다.

`inittab` 파일내의 `sysinit` 라인은 `rc` 나 그 밖의 기본적인 부트스크립트를 구동시킬수 있도록 수정되어야만 합니다. 또한, 시리얼 포트쪽으로 사용자가 접속하는 것을 막으려면 `getty` 설정 엔트리중 라인 끝부분에 `ttys` 나 `ttyS` 디바이스가 적힌 엔트리들은 주석처리 하십시요. 단, 당신이 콘솔로 로그인할 `tty` 포트들 만큼은 남겨두세요.

가장 간단한 `inittab` 파일은 다음과 유사한 모습입니다.

```
id:2:initdefault:
si::sysinit:/etc/rc
1:2345:respawn:/sbin/getty 9600 tty1
2:23:respawn:/sbin/getty 9600 tty2
```

`inittab` 파일은 시스템 기동, 멀티유저 모드로의 이행 등등의 여러 단계에서 시스템이 무엇을 실행시켜야 하는지를 정의한 것입니다. 여기서 세심하게 체크해야할 것은 `inittab` 에서 언급된 화일들이 정말로 제자리에 있는지의 여부입니다. 만일 `init` 가 해당 파일을 찾지 못하면 부트디스크는 멈춰버리게 되며 에러메시지조차 뜨지 않을수도 있습니다.

어떤 프로그램들은 다른 위치에 있는 것이 허용되지 않고 반드시 정해진 디렉토리에 위치해야 합니다. 이는 다른 프로그램 속에 그 위치가 하드코딩되었기 때문입니다. 예를 들어 필자의 시스템의 경우, `/etc/shutdown` 은 `reboot` 의 위치를 `/etc/reboot` 로 하드코딩 하였습니다. 만일 필자가 `reboot` 파일을 `/bin/reboot` 에 둔 후 `shutdown` 명령을 내린다면, `/etc` 디렉토리에서 `reboot` 파일을 찾을 수 없어 실패하고 말 것입니다.

그 밖의 나머지 파일들의 경우, `/etc` 디렉토리의 모든 텍스트 파일들을 복사하시고 만일 뭐가 필요한지 정확히 모르시겠다면 `/etc` 디렉토리의 실행파일들도 모두 복사하십시오. 부록 C 절을 참고하시기 바랍니다. 아마도 거기에 나온 파일들을 복사하는 것으로 충분하겠지만 시스템은 서로 많은 차이가 있을 수 있으므로 당신의 시스템상의 파일들이 견본의 파일들과 같은 역할을 한다고 장담할 수는 없습니다. 가장 확실한 유일한 방법은 `inittab` 에서부터 시작해서 필요한 것들을 하나하나 확인해 나가는 방법 뿐 입니다.

현재 대부분의 시스템들은 각각의 런레벨에 해당하는 쉘 스크립트들을 `/etc/rc.d/` 디렉토리 밑에 두고 있습니다. 필요한 최소한의 파일은 `rc` 스크립트 하나지만, 당신의 원래 시스템

으로부터 일단 inittab 와 /etc/rc.d 디렉토리를 통째로 복사해온 후 디스켓 시스템에 필요 없는 rc.d 디렉토리의 쉘 스크립트들을 하나씩 지워나가는 방법이 더 편리할 것입니다.

4.3.3. /bin 과 /sbin

/bin 디렉토리는 기본적인 작업에 필요한 **ls**, **mv**, **cat**, **dd** 등등의 추가적인 유틸리티들을 두기에 편리한 곳입니다. 부록의 부록 C 에 있는 /bin 과 /sbin 디렉토리의 파일들을 참고하세요. **cpio**, **tar**, **gzip** 등과 같은 백업에 필요한 유틸리티들은 이 디렉토리에 포함시키지 않았습니다. 필자의 경우, 그런 유틸리티들은 boot/root 디스켓의 용량을 아끼기 위해 따로 유틸리티 디스크에 넣어둡니다. 일단 boot/root 디스켓이 부팅이 되어 램디스크로 로딩되고나면, 디스켓을 빼고 유틸리티 디스켓으로 바꿔넣은 후 이를 마운트 할 수 있습니다. 필자는 보통 이 유틸리티 디스켓을 /usr 로 마운트합니다.

유틸리티 디스켓을 만드는 방법은 아래의 8.3절 편에 나와있습니다. 백업을 할 때에는 백업본 외에도 백업을 만드는데 사용된 백업 유틸리티들 역시 동일 버전으로 하나 복사해두는 편이 바람직합니다. 이렇게 해두면 나중에 최신 백업 유틸리티들이 버전의 차이로 인해 옛날 백업 테이프를 읽지 못하는 불상사를 피할 수 있습니다.

> 중요**:** 다음 프로그램들이 포함되었는지 확인하세요: init, getty 류의 프로그램, login, mount, rc 스크립트를 실행시킬 수 있는 쉘 프로그램, 그리고 쉘을 sh 에 링크시켰는지도 확인하십시요.

4.3.4. /lib

/lib 에는 필요한 공유 라이브러리와 로더들을 두어야 합니다. 만일 필요한 라이브러리가 /lib 디렉토리에서 발견되지 않는다면 시스템은 부팅에 실패하게 됩니다. 운이 좋다면 왜 에러가 났는가하는 에러메시지 정도는 받을 수 있을지 모릅니다.

거의 모든 프로그램들이 적어도 libc 라이브러리인 libc.so.**N** 만큼은 반드시 필요로 합니다. 여기서 **N** 은 현재의 버전넘버를 뜻합니다. 당신의 /lib 디렉토리를 확인하세요. 보통, libc.so.N 은 완전한 버전넘버를 가진 파일이름에 심볼릭 링크되어 있습니다.

```
% ls -l /lib/libc.so*
  -rwxr-xr-x   1 root     root      4016683 Apr 16 18:48 libc-2.1.1.so*
  lrwxrwxrwx   1 root     root           13 Apr 10 12:25 libc.so.6 -> libc-2.1.1.so*
```

이 경우, 당신은 libc-2.1.1.so 가 필요합니다. 포함시키려고 하는 바이너리들이 어떤 라이브러리를 필요로 하고 있는지 그 의존성을 검사해 보려면 **ldd** 명령어를 ㅆ십시오. 예를 들면 다음과 같습니다.

```
% ldd /sbin/mke2fs
libext2fs.so.2 => /lib/libext2fs.so.2 (0x40014000)
libcom_err.so.2 => /lib/libcom_err.so.2 (0x40026000)
libuuid.so.1 => /lib/libuuid.so.1 (0x40028000)
libc.so.6 => /lib/libc.so.6 (0x4002c000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

오른편의 각 파일들이 필요함을 알 수 있습니다. 출력된 라이브러리들은 심볼릭 링크일 수 있습니다.

일부 라이브러리들은 상당히 커서 당신의 루트 파일시스템에 쉽사리 들어가지 않을지도 모릅니다. 예를 들어 위에 나온 `libc.so` 는 약 4 메가나 됩니다. 이런 라이브러리들을 루트 화일 시스템으로 옮기려면 스트립(strip)시킬 필요가 있습니다. 스트립시키는 방법은 8.1절 절을 참조하세요.

또한, `/lib` 에는 라이브러리용의 로더를 포함시켜야 합니다. 그 로더는 `ld.so` (A.OUT 라이브러리용으로, 현재 별로 사용되지 않음)이나 `ld-linux.so` (ELF 라이브러리용)일 것입니다. 최근 버전의 **ldd** 는 위의 예처럼 정확히 어떤 로더가 필요한지를 가르쳐주지만 옛날 버전은 그렇지 않습니다. 어떤 로더가 필요한지 자신이 없다면 라이브러리에 대해 **file** 명령을 실행시키세요. 예를 들면 다음과 같습니다.

```
% file/lib/libc.so.4.7.2 /lib/libc.so.5.4.33 /lib/libc-2.1.1.so
/lib/libc.so.4.7.2: Linux/i386 demand-paged executable (QMAGIC), stripped
/lib/libc.so.5.4.33: ELF 32-bit LSB shared object, Intel 80386, version 1, stripped
/lib/libc-2.1.1.so: ELF 32-bit LSB shared object, Intel 80386, version 1, not stripped
```

`QMAGIC` 은 `4.7.2` 가 A.OUT 라이브러리용이고, `ELF` 는 `5.4.33` 과 `2.1.1` 이 ELF 라이브 러리용임을 나타내고 있습니다.

만들고자 하는 루트 파일시스템에 필요한 로더들을 골라 복사하세요. 라이브러리와 로더들이 과연 바이너리에 맞는 것인지 주의깊게 체크해 보아야만 합니다. 만일 커널이 필요한 라이브러리를 로드하지 못하면 대부분의 경우 에러메시지조차 없이 그냥 멈추어 버립니다.

4.4. PAM 과 NSS 에 대한 대책

당신 시스템에는 `ldd` 로 확인할 수 없는 동적 라이브러리가 필요할 수도 있습니다. 그런 경우를 무시했다가는 부트디스크로 로그인하거나 사용할 때 문제가 생길 수 있습니다.

4.4.1. PAM (Pluggable Authentication Modules)

만일 당신의 시스템이 PAM(Pluggable Authentication Modules)을 쓰고 있다면 부트디스크 상에 PAM 을 위한 몇가지 준비를 해주어야 합니다. 간단히 말해서 PAM 이란 사용자를 인증하고 그 사용자의 서비스에 대한 액세스를 컨트롤하는 정교하게 모듈화된 방법입니다. 시스템이 PAM 을 쓰고있는지 쉽게 확인해보려면 `login` 실행화일에 ldd 를 실행시켜 보십시요. `libpam.so` 등의 출력이 나오다면 PAM 이 필요한 것입니다.

운좋게도, 부트디스크에 있어서 보안은 보통 관심밖의 사항입니다. 머신에 이런 식의 물리적 액세스를 할 권한이 있다면 무슨 수를 써서든 소기의 목적한 바를 달성할 수 있을테니까요. 따라서, 루트디스켓에 다음과 비슷한 형태의 간단한 `/etc/pam.conf` 파일을 만들어두면 쉽게 PAM 기능을 무력화시킬 수 있습니다.

```
OTHER   auth        optional     /lib/security/pam_permit.so
OTHER   account     optional     /lib/security/pam_permit.so
```

```
OTHER   password   optional     /lib/security/pam_permit.so
OTHER   session    optional     /lib/security/pam_permit.so
```

또한, `/lib/security/pam_permit.so` 파일을 루트 파일시스템으로 복사하십시오. 이 라이브러리는 겨우 8K 정도이므로 별로 부담스럽지 않습니다.

위와같이 설정하면 이 디스켓으로 당신 머신의 파일이나 서비스에 누구든 아무 제한없이 액세스할 수 있게됩니다. 만일 어떤 이유로 부트디스크상의 보안에도 신경을 써야 하는 상황이라면, 하드디스크의 PAM 설정의 일부 혹은 전부를 당신의 루트 파일시스템으로 복사해야만 합니다. PAM 에 관한 문서를 주의깊게 읽어본 다음 `/lib/security` 에서 필요한 라이브러리들을 루트 파일시스템으로 복사하십시오.

또한 `/lib/libpam.so` 를 부트디스크에 포함시켜야만 합니다. 앞에서 `/bin/login` 에 ldd 를 실행시켰을 적에 이미 이 의존성을 눈치채셨을 것입니다.

4.4.2. NSS (Name Service Switch)

만일 glibc(일명 libc6)를 사용하고 있다면 name service 에 대한 준비를 해주어야만 합니다. 그러지않으면 로그인이 불가능할 것입니다. 파일 `/etc/nsswich.conf` 는 여러가지 서비스에 대한 데이터베이스 열람을 컨트롤합니다. 만일 네트웍상의 서비스(예를 들면 DNS, NIS lookup 등)에 액세스할 필요가 없다면 다음과 같은 간단한 `nsswitch.conf` 파일만 준비하면 됩니다.

```
passwd:     files
shadow:     files
group:      files
hosts:      files
services:   files
networks:   files
protocols:  files
rpc:        files
ethers:     files
netmasks:   files
bootparams: files
automount:  files
aliases:    files
netgroup:   files
publickey:  files
```

이것은 모든 서비스가 오로지 로컬 파일에서 제공되는 것으로 설정한 것입니다. 또한 `/lib/libnss_files.so.`**X** 도 포함시켜야 합니다. 여기서 `X` 는 glibc2.0 에서는 1 이고 glibc2.1 에서는 2 가 됩니다. 이것은 파일 열람(file lookup)을 처리할 때 동적으로 로드되는 라이브러리입니다.

부트디스크에서 네트웍에 액세스할 작정이라면 보다 정교한 `nsswitch.conf` 파일을 만들 필요가 있습니다. 자세한 것은 `nsswitch` 맨 페이지를 참고하세요. 당신이 설정한 **service** 들에 대해, 각각에 해당하는 `/lib/libnss_`**service**`.so.1` 파일들을 포함시켜야만 한다는 점을 명심하십시오.

4.5. 모듈

모듈화된 커널을 사용한다면 부팅 후 부트디스크로부터 어떤 모듈을 로드해야할지를 고려해야만 합니다. 만약 백업 테이프들이 플로피 테이프상에 있다면 **ftape** 와 **zftape** 모듈을 포함시켜야 하고 SCSI 장비를 가지고 있다면 SCSI 관련 모듈을 포함시켜야 하며 만일 응급상황하에서 네트웍에 액세스해야 한다면 PPP 나 SLIP 관련 모듈을 포함시켜야 합니다.

이러한 모듈들은 `/lib/modules` 에 두면 됩니다. 당신은 또 **insmod**, **rmmod**, **lsmod** 프로그램을 포함시켜야 합니다. 모듈을 자동으로 로드하고싶다면 **modprobe**, **depmod**, **swapout** 도 포함시켜야 합니다. **kerneld** 를 사용한다면 **kerneld** 와 그 설정화일인 `/etc/conf.modules` 도 포함시켜야 합니다.

하지만, 모듈을 사용함으로써 얻는 주된 이점은 상대적으로 덜 중요한 모듈들을 유틸리티 디스크에 넣어버리고 필요할 때만 로드함으로써 루트디스크의 공간을 절약하는데 있습니다. 많은 디바이스들을 다루어야 하는 상황이라면 모듈을 이용하는 것이 자체에 많은 드라이버를 내장한 거대한 단일 커널을 쓰는 것보다 더 바람직한 방법입니다.

> 중요**:** 압축된 ext2 파일 시스템을 부트하기 위해서는 램디스크와 ext2 에 대한 지원을 반드시 커널에 내장시켜야만 합니다.이 두가지는 모듈로 설정해서는 절대 안됩니다**.**

4.6. 마지막 세부사항들

login 같은 일부 시스템 프로그램들은 `/var/run/utmp` 파일과 `/var/log` 디렉토리가 없는 경우 문제를 일으킬 수 있습니다. 따라서 다음과 같이 해주십시요.

```
mkdir -p /mnt/var/{log,run}
touch /mnt/var/run/utmp
```

마지막으로, 필요한 모든 라이브러리들을 다 설치했다면 **ldconfig** 를 실행시켜서 루트 파일 시스템 상의 `/etc/ld.so.cache` 를 리메이크 해주십시오. 캐쉬는 로더에게 어디서 라이브러리를 찾아야 할지를 지시합니다. 다음과 같이 하면 됩니다.

```
ldconfig -r /mnt
```

4.7. 만들어진 파일시스템을 포장하기

일단 루트 파일시스템을 다 만들었다면 언마운트시키고 파일로 복사한 다음 압축시켜야 합니다.

```
umount /mnt
dd if=DEVICE bs=1k | gzip -v9 > rootfs.gz
```

이 과정이 끝나면 `rootfs.gz` 라는 파일을 얻게 될텐데 바로 이것이 당신의 압축 루트 파일시스템입니다. 그 크기를 확인해서 과연 한 장의 플로피 디스켓에 다 들어가는지를 체크합니다; 만일 다 들어가지 않는다면 되돌아가서 몇 가지 파일들을 지워야 합니다. 8.1절 부분에 루트 파일시스템의 크기를 줄이는 몇가지 힌트가 있습니다.

5. 커널을 선택하기

이제 당신은 완벽한 압축 루트 파일시스템을 만들었습니다. 다음 과정은 커널을 만드는 것입니다. 대부분의 경우, 현재 시스템의 커널을 그대로 부트디스켓으로 복사해서 그걸로 부트해도 되지만, 때에 따라서는 별도의 커널을 만들어야만 할 때도 있습니다.

커널을 별도로 만드는 이유중 하나는 커널의 크기 때문입니다. 만일 한 장짜리 boot/root 디스켓을 만들려 한다면 커널은 디스켓 상에서 가장 큰 파일이 되기 때문에 가능한한 그 크기를 줄일 필요가 있습니다. 커널 크기를 줄이려면 커널에 당신이 만들려는 시스템에 필요한 최소한의 기능만을 담아야 합니다. 이는 필요없는 기능은 모두 빼라는 뜻입니다. 네트워킹이 그 좋은 예입니다. boot/root 시스템을 운용하는 데 불필요한 다른 디스크 드라이브에 대한 지원도 빼 버리십시오. 기타 불필요한 디바이스들도 버리는 것이 좋습니다. 앞에서도 말했듯이 압축된 ext2 파일 시스템을 부트하기 위해서는 램디스크와 ext2 에 대한 지원만큼은 반드시 커널안에 포함시켜야 합니다.

커널에 포함될 최소한의 기능들을 선정했다면, 이제 다시 무엇을 추가할 수 있을지 확인해 보십시오. 아마도 boot/root 디스켓시스템의 주된 용도는 손상된 루트 파일시스템을 조사하고 복구하기 위한 것일테니 이를 위한 커널차원에서의 지원이 필요합니다. 예를 들어 백업 파일들이 모두 테이프에 담겨있고, 테이프 드라이브를 액세스하기 위해서는 Ftape 를 써야만 하는 시스템이라고 합시다. 만약 Ftape 를 포함하고 있는 시스템의 루트드라이브에 문제가 발생했다면 이제는 백업테이프를 써서 복구할 방법이 없습니다. 결국, 리눅스를 새로 설치한 다음, **ftape** 를 구해 다시 인스톨한 후에야 백업 테이프를 읽어올 수 있다는 말이 됩니다.

이 예에서 지적하고자 하는 바는, 시스템 커널에 있는 백업장비에 대한 I/O 지원은 boot/root 디스켓의 커널에도 그대로 포함시켜야 한다는 것입니다.

커널을 만드는 절차는 커널소스에 동봉된 문서에 나와있습니다. 보면 쉽게 따라할 수 있게 되어있으므로 `/usr/src/linux` 에서 내용을 읽어 보십시요. 커널을 제대로 만들지 못한다면 boot/root 시스템을 만들 수도 없습니다. 압축커널을 만들 때는 "**make zImage**" 명령을 사용해야함을 기억하세요.

6. 만든 것들을 하나로 모으기 : 디스켓 제작

당신은 이제 커널과 압축 루트 파일시스템을 가지고 있습니다. 한장짜리 boot/root 디스크를 만들겠다면 커널과 압축 루트 파일시스템을 합친 용량이 한 장의 디스켓에 다 들어가는지 확인해 보십시요. 만일 두장짜리 boot+root 디스크를 만든다면 루트 파일시스템이 한장의 디스켓에 다 들어가는지 확인해 보십시요.

또한, 부트디스크의 커널을 부트시키는데 있어 부트로더의 일종인 LILO 를 사용할 지 여부를 결정해야 합니다. 다른 방법으로는 LILO 를 쓰지 않고 커널을 직접 디스켓에 카피해서 그 디스켓으로 부팅하는 방법도 있습니다. LILO 를 쓰면 하드웨어를 어떻게 초기화시킬지에 대한 파리메터를 부팅시에 커널에 지시할 수 있다는 장점이 있습니다(당신 시스템의 /etc/lilo.conf 파일을 체크해 보십시오. 만일 이 파일이 있고 그 안에 "append=..." 하는 라인이 있다면 당신은 이미 이 기능을 쓰고있는 것입니다). LILO 를 썼을 때의 단점은 부트디스크를 만드는 과정이 더 복잡해지고 용량도 조금 더 차지한다는 것입니다. LILO 를 쓰자면 소위 커널 파일시스템이라 불리우는 별도의 작은 파일 시스템을 만들어 거기에다 커널과 그 밖에 LILO 가 필요로 하는 몇몇 파일들을 담아야만 합니다.

LILO 를 사용하겠다면 계속 읽어나가시고 직접 커널을 전송하는 방법을 택하겠다면 6.2절 부분으로 건너뛰시기 바랍니다.

6.1. 디스켓에 LILO 와 커널을 함께 담기

맨 먼저 할 일은 LILO 를 위한 작은 설정파일을 만드는 것입니다. 다음과 비슷한 내용이 됩니다.

```
boot      =/dev/fd0
install   =/boot/boot.b
map       =/boot/map
read-write
backup    =/dev/null
compact
image     = KERNEL
label     = Bootdisk
root      =/dev/fd0
```

이들 파라메터에 대해서는 LILO 사용자 설명서를 참조하십시오. 또한 당신의 하드디스크의 /etc/lilo.conf 파일에 적혀있는 append = ... 라인을 보고 위의 화일 끝에 그대로 추가해야할 것입니다.

이 파일을 bdlilo.conf 라는 이름으로 저장하십시오.

이제 커널 파일시스템이라 불리우는 작은 파일시스템을 만들어야만 합니다. 이것은 루트 파일시스템과는 별개의 것입니다.

먼저, 파일시스템의 크기를 얼마로 해야할지 알아내야 합니다. 만들어낸 커널의 블록 단위 크기가 얼마인지 확인하고("**ls -s KERNEL**" 명령으로 알수 있습니다), 거기에 50 을 더합니다. 50 블록은 inode 와 그 밖의 몇 가지 파일들에 필요한 대략적인 크기입니다. 원하신다면 정확히 계산해 볼 수도 있습니다. 아니면 그냥 50 을 사용하세요. 두장의 디스켓으로 만든다면 첫번째 디스크는 어쨌든 커널전용으로만 사용될테니 이 크기를 더 넉넉하게 잡아도 좋습니다. 계산한 숫자를 KERNEL_BLOCKS 라고 부르기로 합시다.

플로피 디스켓을 드라이브에 넣고 그 위에 ext2 커널 파일시스템을 만듭시다(편의상 드라이브의 이름은 /dev/fd0라 가정합니다).

```
mke2fs -N 24 -m 0 /dev/fd0 KERNEL_BLOCKS
```

"-N 24 " 는 inode 수를 24 개로 설정합니다. 이 파일시스템에 필요한 inode 수는 24 개면 됩니다.그 다음, 이 파일시스템을 마운트하고 lost+found 디렉토리를 삭제한 후 LILO 를 위한 dev 와 boot 디렉토리를 만듭니다.

```
mount /dev/fd0 /mnt
rm -rf /mnt/lost+found
mkdir /mnt/{boot,dev}
```

그 다음, 디바이스 /dev/null 과 /dev/fd0 를 만듭니다. 디바이스 넘버를 찾는 대신 그냥 당신 시스템의 하드디스크로부터 -R 옵션을 주어 복사해오면 됩니다.

```
cp -R /dev/{null,fd0} /mnt/dev
```

LILO 는 그 부트로더인 boot.b 의 복사본을 필요로 하는데, 이 boot.b 는 하드디스크에서 가져오면 됩니다. 보통, 이 파일은 /boot 디렉토리에 있습니다.

```
cp /boot/boot.b /mnt/boot
```

마지막으로, 앞 절에서 만든 LILO 설정파일과 커널을 플로피로 복사해 넣습니다. 두 가지 모두 루트 디렉토리에 들어갑니다.

```
cp bdlilo.conf KERNEL /mnt
```

이제 LILO 가 필요로 하는 모든 것들이 커널 파일시스템에 들어갔으므로 준비는 모두 끝났습니다. LILO 의 -r 플래그는 부트로더를 다른 루트에 설치할 때 사용됩니다.

```
lilo -v -C bdlilo.conf -r /mnt
```

LILO가 아무 에러도 없이 제대로 동작했다면 커널 파일시스템은 다음과 비슷한 모습이 될 것입니다.

```
total 361
  1 -rw-r--r--   1 root     root          176 Jan 10 07:22 bdlilo.conf
  1 drwxr-xr-x   2 root     root         1024 Jan 10 07:23 boot/
  1 drwxr-xr-x   2 root     root         1024 Jan 10 07:22 dev/
358 -rw-r--r--   1 root     root       362707 Jan 10 07:23 vmlinuz
boot:
total 8
  4 -rw-r--r--   1 root     root         3708 Jan 10 07:22 boot.b
  4 -rw-------   1 root     root         3584 Jan 10 07:23 map
dev:
total 0
  0 brw-r-----   1 root     root       2,   0 Jan 10 07:22 fd0
  0 crw-r--r--   1 root     root       1,   3 Jan 10 07:22 null
```

당신이 만든 것과는 파일 크기가 약간 틀릴 수도 있으니 걱정하지 마십시요.

이제 드라이브의 디스켓은 그대로 두고 6.3절 편으로 가시기 바랍니다.

6.2. LILO 없이 커널만 담기

LILO 를 사용하지 않겠다면 **dd** 명령을 써서 커널을 부트디스크에 담아야 합니다.

```
% dd if=KERNEL of=/dev/fd0 bs=1k
353+1 records in
353+1 records out
```

위의 예에서 **dd** 는 353 개의 완전한 레코드와 1개의 부분레코드를 써넣었습니다. 따라서 커널은 디스켓 앞부분의 354 블록들을 점유하게 됩니다. 이 숫자를 `KERNEL_BLOCKS` 라고 부르겠으니 기억해 두십시오. 다음 절에서 사용할 것입니다.

마지막으로 루트디바이스를 디스켓 자체로 설정한 후, 루트가 읽기/쓰기가 가능하게 로드되도록 설정합시다.

```
rdev /dev/fd0 /dev/fd0
rdev -R /dev/fd0 0
```

두 번째 줄의 **rdev** 명령에서 대문자 `-R` 을 써야함에 주의하세요.

6.3. 램디스크 워드의 설정

커널 이미지 내에는 램디스크 워드라는 것이 있습니다. 이것은 다른 옵션들과 더불어서 루트 파일시스템을 어디에서 찾을 것인지를 설정합니다. 이 워드는 **rdev** 명령을 써서 확인 및 설정이 가능합니다. 램디스크 워드의 내용은 다음과 같이 해석합니다.

Bit field	의미
0-10	1024 byte 블록기준, 램디스크가 시작하는 오프셋
11-13	사용 않음
14	램디스크로 로딩될지를 나타내는 플래그
15	루트 파일시스템을 로드하기 직전에 한번 멈출지를 결정하는 플래그

만약 15번 비트가 설정되어 있으면 부팅시에 새로운 디스켓을 집어넣으라는 메시지를 받게 될 것입니다. 이 기능은 두 개의 디스크로 부팅할 경우 필요합니다.

한 개의 boot/root 디스켓을 만들 것인지, 아니면 두개의 "boot+root" 디스켓 세트를 만들 것인지에 따라 다음 두 가지 경우가 생길 수 있습니다.

1. 한 개의 디스켓를 만든다면 압축 루트파일 시스템은 커널 바로 뒤에 연이어 위치하게 되므로 그 오프셋은 빈 블럭의 첫번째(`KERNEL_BLOCKS` 값과 동일)가 됩니다. 비트 14 는 1 로, 비트 15 는 0 으로 설정해야 합니다.

   예를 들어, 당신이 한장짜리 디스크를 만드는데 루트 파일시스템이 253 블록(십진수)에서 시작한다고 칩시다. 램디스크 워드의 값은 253(십진수)에다 비트 14 는 1, 비트 15 는 0

으로 세팅한 값입니다. 이 램디스크 워드 값을 구하려면 단순히 모두 십진수로 변환해 더하면 됩니다. 253 + (2^14) = 253 + 16384 = 16637 입니다. 이 값이 어디서 온 건지 아무래도 이해를 못시겠다면 전자계산기를 써서 이 값을 이진수로 변환해 보시면 이해가 가실 것입니다.

2. 두개의 디스켓 세트를 만든다면 루트 파일시스템은 두 번째 디스켓의 0 번 블록부터 시작할 것이고 따라서 그 오프셋은 0 입니다. 비트 14는 1 로, 비트 15 도 1 로 설정합니다. 따라서 램디스크 워드의 십진수 값은 이 경우 2^14 + 2^15 = 49152 가 됩니다.

램디스크 워드에 해당하는 값을 주의깊게 계산한 다음, 그 값대로 **rdev -r** 명령으로 설정합니다. 십진수를 사용해야 함에 주의하십시요. LILO를 사용하겠다면 마운트될 커널의 경로, 예를 들면 `/mnt/vmlinuz` 같은 것을 **rdev** 명령어에 파라메터로 써주어야 합니다. LILO 를 쓰지 않고 커널을 직접 **dd** 명령으로 복사하겠다면 대신 플로피 디바이스의 이름을 써줍니다 (예를들면 `/dev/fd0`).

```
rdev -r KERNEL_OR_FLOPPY_DRIVE  VALUE
```

LILO를 사용한다면 이제 디스켓을 언마운트 시키십시오.

6.4. 루트 파일시스템을 디스켓에 담기

마지막 단계는 루트 파일시스템을 플로피 디스켓에 담는 것입니다.

- 루트 파일시스템을 커널과 같은 디스켓에 담는다면, **dd** 명령에 `seek` 옵션을 주십시오. 이 옵션은 얼마만큼의 블록을 건너뛰어야 하는지를 설정합니다.

  ```
  dd if=rootfs.gz of=/dev/fd0 bs=1k seek=KERNEL_BLOCKS
  ```

- 만일 루트 파일시스템을 두번째 디스켓에 담는다면, 첫번째 디스켓을 빼고 두번째 디스켓을 드라이브에 넣은 후 루트 파일시스템을 담습니다.

  ```
  dd if=rootfs.gz of=/dev/fd0 bs=1k
  ```

축하합니다. 이제 끝났습니다!

> 중요**:** 응급상황용으로 따로 보관해 두기전에 먼저 이 부트디스크가 제대로 동작하는지 확실히 테스트해보시기 바랍니다. 만일 부트에 실패하셨다면 이 문서를 계속 읽어나가십시오.

7. 애로사항과 문제해결

부트디스크를 만들 때 단번에 성공하는 일을 거의 없습니다. 루트디스크를 만드는 일반적인 접근방법은 우선 당신의 현재 시스템으로부터 필요한 요소들을 끌어모아 조립한 후, 시행착

오를 거쳐 콘솔 상에 어떤 메시지가 나타나는 단계에까지 진입하는 것입니다. 일단 디스켓 시스템이 어떤 메시지를 출력하는 단계에까지 이르면 전투의 반은 끝난 것이나 다름없습니다. 왜냐하면 이 단계까지 오면 출력된 메시지를 보고 무엇이 문제인지를 파악해 나갈 수 있으므로 시스템이 원활히 동작할 때까지 각각의 개별적 문제들을 해결해 나가기만 하면 되기 때문입니다. 시스템이 아무런 메시지 없이 그냥 멈추어 버린 경우라면 그 원인을 찾는 것은 어려운 일입니다. 만일 시스템이 아무런 메시지를 남기지 않고 멈추어 버렸다면 아래의 순서대로 원인을 조사해 나가십시요.

- 다음과 같은 메시지가 나온 경우

```
Kernel panic: VFS: Unable to mount root fs on XX:YY
```

이는 흔히 발생하는 문제이며 이유는 몇가지 밖에 없습니다. 우선 디바이스 **XX:YY** 와 `/usr/src/linux/Documentation/devices.txt` 에 나와있는 디바이스 코드를 대조해보세요. 그것이 올바른 디바이스가 아니라면 원인은 틀림없이 **rdev -R** 을 실행하지 않았거나 잘못된 이미지에 대고 rdev 명령을 내린 것입니다. 디바이스 코드가 올바르다면 당신의 커널내에 컴파일되어있는 디바이스 드라이버들을 주의깊게 체크해 보세요. 플로피디스크, 램디스크, ext2 파일시스템에 대한 지원이 모듈이 아닌 커널에 직접 내장되었는지 확인하십시요.

- 다음과 같은 에러들이 줄줄이 나올때 :

```
end_request: I/O error, dev 01:00 (ramdisk), sector NNN
```

이것은 램디스크 드라이버가 발생시킨 I/O 에러입니다. 주로 커널이 디바이스 용량을 초과해 기록하려할 때 발생합니다. 램디스크가 너무 작아서 루트 파일시스템을 다 담을수 없는 것입니다. 부트 디스크의 부팅시에 나타나는 커널 초기화 메시지에 다음과 유사한 줄이 있는지 확인하세요.

```
Ramdisk driver initialized : 16 ramdisks of 4096K size
```

이 크기와 압축되지 않은 상태의 루트 파일시스템의 크기를 대조해 보세요. 램디스크가 이보다 충분히 크지 않다면 더 크게 잡아주세요.

- 루트디스크에 정말로 당신이 생각했던 디렉토리들이 포함되어 있는지 확인합니다. 착각하는 바람에 루트디스켓의 `/bin` 을 만드는 대신 `/rootdisk/bin` 을 만드는 식의 실수를 하기 쉽습니다.
- 루트 파일시스템의 `/lib/libc.so` 의 링크가 하드디스크의 `/lib` 디렉토리에 있는 링크와 같은지 확인합니다.
- 루트디스켓 파일시스템의 `/dev` 디렉토리의 심볼릭 링크가 당신 시스템의 그것과 동일한지 확인합니다. 디바이스에 대한 심볼릭 링크들은 루트 디스켓에서도 그대로 포함되어야 합니다. 특히 `/dev/console` 링크들은 대부분의 경우 반드시 있어야 합니다.
- `/dev/tty1, /dev/null, /dev/zero, /dev/mem, /dev/ram, /dev/kmem` 파일들이 포함되었는지 확인합니다.
- 커널 설정을 확인합니다. 로그인 단계에 다다를 때까지에 필요한 모든 자원들에 대한 지원은 커널에 내장되어야지 모듈로 설정되어서는 안됩니다. 따라서 램디스크와 **ext2** 에 대한 지원은 반드시 커널에 내장되어야만 합니다.

- 커널 루트 디바이스와 램디스크 설정이 올바른지 확인합니다.

일단 위의 일반적인 사항들을 확인했다면 이제 보다 구체적인 파일들을 확인합니다.

1. **init** 가 `/sbin/init` 혹은 `/bin/init` 로 제대로 포함되었는지 확인합니다. 실행가능한 상태인지도 확인합니다.
2. **ldd init** 해서 init 의 라이브러리들을 체크합니다. 보통 이것은 `libc.so` 가 되지만 하여튼 확인합니다. 필요한 라이브러리와 로더들을 포함시켰는지 확인합니다.
3. 각 라이브러리들에 대해 그에 해당하는 알맞은 로더를 가지고 있는지를 확인합니다. a.out 에는 `ld.so` 가 있어야 하고 ELF 에는 `ld-linux.so` 가 있어야 합니다.
4. 부트디스크 파일시스템의 `/etc/inittab` 파일에 있는 **getty**(혹은 **agetty**, **mgetty**, **getty_ps** 등의 이른바 **getty** 류 프로그램)를 호출하는 부분을 체크합니다. 이 부분을 하드 디스크의 `inittab` 과 비교하면서 되풀이하여 확인해 봅니다. 맨 페이지를 펼쳐놓고 과연 제대로 설정되었는지 확인합니다. `inittab` 는 리눅스 시스템에서 가장 교묘한 부분입니다. 이유는 그 문법과 내용이 사용되는 init 프로그램에 따라 서로 다르고, 또 각 시스템마다 조금씩 다르기 때문입니다. 이에 관한 문제를 다루는 유일한 방법은 **init** 와 `inittab` 에 대한 맨 페이지를 숙지한 후, 당신의 시스템 본체가 부트될 때 일어나는 과정들이 플로피 디스크 상에서도 똑같이 일어나도록 하는 것입니다. `/etc/inittab` 가 시스템 초기화 엔트리를 가지고 있는지 확인하십시요. 이 파일에는 시스템 초기화 스크립트들을 수행시키는 명령어가 반드시 포함되어 있어야만 합니다.
5. **init** 에 했던 것처럼 **getty** 에 대해서도 **ldd** 를 실행시켜서 **getty** 가 무엇을 필요로 하는지 확인하고, 필요한 라이브러리와 로더들이 루트 파일시스템에 들어있는지 확인합니다.
6. 쉘 프로그램(예를들면 **bash** 나 **ash** 등등)을 포함시켰는지 확인하고, 이 쉘 프로그램들이 rc 스크립트들을 과연 제대로 실행시킬수 있는지 확인합니다.
7. 만일 복구디스켓에 `/etc/ld.so.cache` 파일을 포함시켰다면 그것을 리메이크합니다.

init 가 시작되기는 하는데 다음과 같은 메시지를 내는 경우 :

```
Id xxx respawning too fast: disabled for 5 minutes
```

이것은 **init** 가 발생시킨 메시지로서 주로 **getty** 나 **login** 이 시작하자마자 죽어버렸음을 알리는 것입니다. **getty**, **login** 실행화일과 이들이 의존하는 라이브러리들을 확인해 보십시오. `/etc/inittab` 내의 호출형식이 정확한지 확인하십시오. 만일 **getty** 로부터 이상한 메시지를 받았다면 아마도 `/etc/inittab` 내의 호출형식이 어딘가 틀렸을 것입니다.

만일 로그인 프롬프트가 떴고 사용자 이름을 제대로 입력했는데도 시스템 프롬프트가 즉각 또다른 로그인 네임을 요구한다면, 문제는 아마도 PAM 이나 NSS 에 관련된 것일 겁니다. 4.4절 절을 참고하세요. 또한, shadow password 를 사용하면서도 깜박 `/etc/shadow` 를 부트디스크로 복사해 넣지 않았기 때문일 수도 있습니다.

복구 디스켓에 있는 **df** 등의 일부 실행파일을 실행했을때 다음과 비슷한 메시지를 받는 경우: `df: not found`, 다음 두가지를 확인하십시요. (1) 그 바이너리가 위치한 디렉토리가 PATH 에 잡혀있는지 (2) 그 프로그램이 필요로 하는 라이브러리와 로더를 포함시켰는지.

8. 그밖의 주제들

8.1. 루트 파일시스템의 크기를 줄이는 방법

때로는 루트 파일시스템이 너무 커서 압축을 해도 디스켓 한장에 다 들어가지 않을 수 있습니다. 다음은 파일 시스템의 크기를 줄이는 방법입니다.

1. 디스크의 밀도를 높입니다. 디폴트 값으로 플로피 디스켓은 1440 K 로 포맷됩니다, 하지만 더 높은 밀도의 포맷도 가능합니다. **fdformat** 명령어는 디스크를 다음과 같은 크기로 포맷할 수 있습니다: 1600, 1680, 1722, 1743, 1760, 1840, 1920. 대부분의 1440 K 드라이브들은 1722 K 의 포맷을 지원하며 필자도 언제나 부트디스크로 이 포맷을 쓰고 있습니다. **fdformat** 의 맨 페이지와 `/usr/src/linux/Documentation/devices.txt` 를 참조하십시오.
2. 쉘을 바꿉니다. 리눅스에서 인기 있는 쉘은 **bash**, **tcsh** 등등이 있지만 이것들은 크기도 크고 많은 라이브러리들을 필요로 합니다. **ash**, **lsh**, **kiss**, **smash** 같은 경량급 쉘들은 훨씬 작고 라이브러리를 별로 필요로 하지 않거나 전혀 요구하지 않으므로 대안이 될 수 있습니다. 이러한 대용 쉘들은 대부분 다음 홈페이지에서 찾을 수 있습니다. `http://metalab.unc.edu/pub/Linux/system/shells/`. 어떤 쉘을 쓰든 간에 그 쉘은 부트디스크에 포함시킨 모든 `rc` 파일들 내의 명령어들을 실행시킬 수 있어야 합니다.
3. 스트립 라이브러리와 바이너리들을 사용합니다. 많은 라이브러리와 바이너리들이 디버깅 정보를 포함한 채 배포됩니다. 이런 파일들에 대해 **file** 명령을 실행하면 "`not stripped`" 라는 결과가 출력됩니다. 바이너리들을 루트 파일시스템으로 복사할 때는 다음과 같은 습관을 들이는 것이 좋습니다.

   ```
   objcopy --strip-all FROM TO
   ```

   중요**:** 라이브러리를 복사할 때는 `strip-all` 대신 `strip-debug` 을 사용하세요.
4. 루트 파일시스템을 만들면서 화일의 삭제, 이동이 많았다면 새로 만드십시요. 위에서 말했듯이 파일 시스템상의 블록들을 깨끗하게 유지하는 것이 중요합니다.
5. 핵심적이지 않은 파일들은 유틸리티 디스크로 옮깁니다. 부트나 로그인 시에 반드시 필요한 것이 아니라면 그런 바이너리들은 유틸리티 디스크로 옮겨놓습니다. 자세한 것은 8.3절 절을 보십시오. 모듈들을 유틸리티 디스크로 옮겨놓는 것도 고려해 볼만 합니다.

8.2. 램디스크 아닌 루트 파일시스템

4절 편에서는 시스템 부팅과 동시에 램디스크로 로드되는 압축 루트 파일시스템 제작법을 설명했습니다. 이 방법이 많은 장점이 있어 주로 사용됩니다만 메모리가 부족한 일부 시스템은 램이 램디스크를 만들만한 용량도 못되는 수가 있으므로 이때는 디스켓 상에서 직접 마운트되는 루트 파일시스템을 만들어야 합니다.

이러한 파일 시스템은 다른 디바이스가 아닌 디스켓 위에 그대로 만들수 있고, 또 압축도 필요없기 때문에 사실 압축 루트 파일시스템을 만드는 것보다 쉽습니다. 위에서 우리가 배운 절차와는 조금 다르므로 그 개요를 적어보겠습니다. 이 방법을 택하면 사용할 수 있는 공간이 훨씬 적어진다는 사실을 잊지 마십시요.

1. 루트파일들에 할당할 수 있는 공간이 얼마나 되는지 계산한다. 만일 한장짜리 boot/root 디스크를 만든다면 커널의 블록과 루트 파일시스템의 블록을 더한 값이 디스켓 한 장의 용량에 맞아야 한다.
2. **mke2fs** 를 써서 디스켓 위에 적절한 크기의 루트 파일시스템을 만든다.
3. 앞에서 배운 대로 파일 시스템을 구성한다.
4. 다 되었으면 파일 시스템을 언마운트시킨 후 디스크파일 한개로 만든다. 단, 압축시키지는 말라.
5. 앞에서 배운대로 커널을 플로피디스켓에 담는다. 램디스크 워드를 계산할 때는 비트 **14** 를 **0** 으로 설정한다. 이는 루트 파일시스템이 램디스크로 로드되지 않도록 하는 것이다. 앞에서 배운 대로 **rdev** 를 실행한다.
6. 앞에서 배운대로 루트 파일시스템을 플로피 디스켓에 담는다.

몇가지 지름길이 있습니다. 만일 두장의 디스크 세트를 만든다면 직접 두번째 디스크 상에 루트 파일시스템을 만들면 됩니다. 굳이 하드디스크 위에서 만들어 옮겨올 필요가 없지요. 또한 한장짜리 boot/root 디스크를 만들면서 LILO를 사용하겠다면, 한장의 디스켓에 단일한 파일시스템을 만들후 여기다가 커널, LILO 에 필요한 파일들, 루트파일들 셋을 모두 집어넣은 후 최후에 LILO 를 실행시켜주면 됩니다.

8.3. 유틸리티 디스크 만들기

유틸리티 디스켓을 만드는 것은 비교적 쉽습니다 -- 그저 포맷된 디스크에 파일 시스템을 만들고 거기에 파일들을 복사하면 됩니다. 부트디스켓에서 이 유틸리티 디스켓를 이용하려면 시스템이 부트된 후 유틸리티 디스켓을 수동으로 마운트하면 됩니다.

이 문서의 앞부분에서 유틸리티 디스켓를 `/usr` 디렉토리에 마운트할 수 있다고 말했습니다. 이 경우 바이너리들은 현재 유틸리티 디스켓 상의 `/bin` 디렉토리 아래에 위치하고 있으므로 `/usr/bin` 을 PATH 에 포함시켜두면 이를 액세스할 수 있습니다. 실행화일에 필요한 각종 라이브러리들은 유틸리티 디스켓의 `/lib` 디렉토리에 두면 됩니다.

유틸리티 디스크 제작시 명심해야할 중요한 사항들이 몇가지 있습니다.

1. 핵심적인 시스템 바이너리나 라이브러리들은 유틸리티 디스크에 담지 마십시요. 유틸리티 디스크는 시스템이 부트된 후에야 마운트될 수 있기 때문입니다.
2. 플로피 디스켓과 플로피 테이프드라이브를 동시에 엑세스할 수는 없습니다. 이 말은 플로피 테이프 드라이브를 가지고 있다해도 유틸리티 디스켓이 마운트 되어있는 동안에는 이 테이프 드라이브를 액세스 할 수 없다는 뜻입니다.

3. 유틸리티 디스켓에 있는 파일을 엑세스하는 속도는 상당히 느립니다.

부록 D 은 유틸리티 디스크에 들어가는 파일들의 예를 보여줍니다. 도움되는 아이디어를 발견할 수 있을 것입니다: 디스크를 다루는 프로그램들(**format, fdisk**)과 파일 시스템용 프로그램들(**mke2fs, fsck, debugfs, isofs.o**), 간단한 텍스트 에디터 (**elvis, jove**), 압축및 아카이브 유틸리티(**gzip, bzip, tar, cpio, afio**), 테이프 유틸리티(**mt, ftmt, tob, taper**), 통신 유틸리티(**ppp.o, slip.o, minicom**), 디바이스용 유틸리티(**setserial, mknod**) 등이 들어있습니다.

9. 전문가들이 사용하는 방법

슬랙웨어, 레드햇, 데비안 등의 주요한 배포본들에 사용되는 부트디스크도 한번쯤 생각해볼 수 있습니다. 이런 것들은 이 문서에서 설명한 것보다 복잡하게 구성되어 있습니다. 전문적인 배포본의 부트디스크들 역시 여기에서 대략 설명한 원리에 기초하고 있습니다만, 그 외에도 그런 부트디스크들은 다음의 기능들을 구비하기 위해 보다 다양한 기교를 사용합니다. 첫째, 폭넓은 종류의 하드웨어를 지원해야 합니다. 따라서 사용자의 입력을 받을수 있어야 하고 다양한 디바이스 드라이버들을 로드할수 있어야 합니다. 둘째, 여러가지 많은 설치 옵션을 입력받아 각각을 자동적으로 처리할 수 있어야 합니다. 마지막으로, 배포본의 부트디스크들은 대개의 경우 배포본의 설치기능과 응급조치의 기능을 함께 가지고 있습니다.

어떤 부트디스크들은 **initrd**(초기 램디스크)라 불리우는 기능을 이용합니다. 이 기능은 커널 2.0.x 대에서 처음 도입되었으며 커널을 두 단계로 부트시킵니다. 일단, 커널이 처음 부트된 후 초기 램디스크 이미지를 부트디스크에서 읽어옵니다. 초기 램디스크 이미지는 진짜 루트 파일시스템이 로드되기에 앞서 먼저 실행되어야할 프로그램들을 담은 루트 파일시스템입니다. 이 프로그램은 시스템 환경을 조사하고 사용자로 하여금 다양한 부트옵션을 선택할 수 있게 해줍니다. 가령 진짜 루트디스크를 어느 디바이스에서 로드할지를 선택할 수도 있습니다. 이것은 주로 커널에 내장되어있지 않은 추가적인 모듈들을 로드합니다. 이 초기화 프로그램이 끝나면 커널은 이제 진짜 루트이미지를 로드해서 정상적으로 부팅을 속개하게됩니다. **initrd** 에 관한 더 많은 내용은 `/usr/src/linux/Documentation/initrd.txt` (file:/usr/src/linux/Documentation/initrd.txt) 와 `ftp://elserv.ffm.fgan.de/pub/linux/loadlin-1.6/initrd-example.tgz`

다음은 각 배포본의 설치 디스크들이 어떤 식으로 작동하는지 파일 시스템들과 소스코드를 기반으로 대강 살펴본 것입니다. 이 내용이 확실한 것인지, 또 각 배포본들이 버전이 올라감에 따라 설정을 바꾸었는지 여부에 대해 장담은 못드립니다.

슬랙웨어(v.3.1)는 6.1절 부분에서 설명한 직관적인 LILO 부트방식과 유사한 방식을 사용합니다. 슬랙웨어의 부트디스크는 LILO 의 `message` 파라메터를 이용하여 부트 업 메시지 (“`Welcome to the Slackware Linux bootkernel disk!`”)를 화면에 출력합니다. 이 메시지는 사용자로 하여금 필요한 경우 부트 파라메터 라인을 입력토록 지시합니다. 부팅 후 루트 파일시스템은 두번째 디스크에서 로드됩니다. 이제 사용자는 초기화과정을 처리하는 **setup** 스크립트를 가동시키게 됩니다. 모듈화된 커널을 쓰는 대신 슬랙웨어는 각각의 커널을 다양하게 준비해 두고 그 중에서 사용자가 자신의 하드웨어 사양에 맞는 것 하나를 골라쓰는 방법을 택하고 있습니다.

레드햇(v.4.0) 역시 LILO 부트를 이용합니다. 레드햇은 첫번째 디스크에서 압축된 램디스크를 로드하며, 이는 레드햇 특유의 **init** 프로그램을 기동시킵니다. 이 프로그램은 드라이버를 물어본 후 필요한 경우 보충 디스크에서 추가적인 파일들을 로드하게 됩니다.

데비안(V.1.3) 은 설치디스크들 중에서 가장 복잡한 방법을 쓰고 있습니다. 이것은 SYSLINUX 로더를 써서 다양한 로드 옵션을 제공한 다음, `initrd` 이미지를 사용해서 설치과정동안 사용자를 안내합니다. 데비안은 데비안 특유의 **init** 와 쉘을 사용하는 듯 합니다.

10. 부팅가능한 CD-ROM 제작

> 참고**:** 이 절은 Rizwan Mohammed Darwe(<`rizwan@clovertechnologies.com`>) 씨가 담당해 주셨습니다.

이 절은 당신이 이미 리눅스에서 CD 를 만드는 작업을 잘 알고 있다고 가정합니다. 이 절을 부팅 CD 를 굽는 간단한 안내서로 생각해 주십시요. CD-Writing-HOWTO 문서에는 더 깊이있는 내용이 실려있습니다.

10.1. 엘 토리토(El torito) 란 무엇인가?

x86 플랫폼의 많은 BIOS 제작사들이 CD 부팅을 지원하기 시작했습니다. mkisofs 에 대한 패치는 엘 토리토 라는 표준에 기반하고 있습니다. 간단히 말해 엘 토리토란 CD 로 직접 부팅하기 위해서는 시디롬이 어떻게 포맷되어야 하는가에 관한 표준 규약입니다.

엘 토리토 규약에는 BIOS 가 엘 토리토를 지원하는 한 어떠한 시디롬으로도 부팅할 수 있다고 되어있습니다만, 지금 현재 SCSI 컨트롤러들중 엘 토리토를 지원하는 것은 전혀 없으며, 단지 EIDE 드라이브들만이 엘 토리토를 지원하고 있습니다. 마더보드가 반드시 엘 토리토를 지원해야 합니다. 자신의 마더보드가 엘 토리토를 지원하는지 어떻게 알수 있냐고요? 엘 토리토가 지원되는 마더보드는 BIOS 설정에서 하드 디스크, 플로피 디스크, 네트웍 또는 시디롬 중 어떤 매체로 부팅할 지를 선택할 수 있습니다.

10.2. 작동 원리

엘 토리토 규약은 BIOS 호출을 이용해서 시디 드라이브를 마치 플로피 드라이브인 양 속여서 동작합니다. 이 방법을 써서 당신은 어떠한 플로피 크기의 이미지(예를 들면 1.44 M 플로피의 경우 1440 Kbyte)라도 ISO 파일 시스템속에 넣어둘 수 있습니다. ISO 파일 시스템의 헤더속에 이 이미지에 대한 포인터를 넣어주면 됩니다. 그러면 BIOS 가 이 이미지를 CD 에서 찾아서 마치 플로피 드라이브로부터 부팅하는 것과 똑같이 부팅하게 됩니다. 한 예로, LILO 부트디스크조차도 똑같이 동작하게 됩니다.

간단히 말해, 시디롬의 첫번째 1.44 MByte(2.88 M 디스크라면 2.88 Mbyte) 부분에 당신이 넣어놓은 플로피 디스크의 이미지가 들어갑니다. 이 이미지는 BIOS 에 의해 플로피로 인식되어져 여기서 부팅이 이루어집니다. (결국, 이 가상의 플로피로 부팅하는 동안은 이게 A: 가 되므

로 원래의 진짜 A:(/dev/fd0) 로는 액세스할 수 없으며, /dev/fd1 를 통해 액세스해야 합니다)

10.3. 제작 방법

먼저 화일을 하나 만듭시다. boot.img 라고 합시다. 이것은 CD-ROM 을 통해 부트하고자 하는 부팅가능한 플로피 디스켓의 이미지입니다. 이것은 반드시 1.44 MB 의 부팅가능한 플로피여야 합니다. 명령은 아래와 같습니다.

```
dd if=/dev/fd0 of=boot.img bs=10k count=144
```

플로피는 A: 드라이브에 있다고 가정했습니다.

이 이미지를 iso9660 화일 시스템의 디렉토리계층 속 어딘가에 넣어둡시다. 부팅에 관련있는 화일들을 모두 한 디렉토리에 모아두는 것이 좋은 방법입니다(예를 들면 iso9660 파일시스템의 루트디렉토리 밑에 boot/ 로 모아두는 방법).

잠깐! -- 당신의 부트 플로피는 반드시 LILO 를 통해서만 초기 램디스크를 로드해야 합니다. 커널 램디스크 드라이버를 사용해서는 안됩니다! 그 이유는 일단 리눅스 커널이 시작되고나면 BIOS 가 CD 를 플로피 디스크로 속였던 것이 더이상 유효하지 않게되어 부팅에 실패하게 되기 때문입니다. LILO 는 BIOS 의 디스크 호출을 통해 초기 램 디스크를 로드하므로, CD 를 계속 플로피로 인식시킬 수 있습니다.

또한, 엘 토리토 규약에는 "부트 카탈로그" 라는 것을 만들어야 한다고 되어있습니다. 이것은 2048 byte 의 파일로서, 어떤 기능을 하는 것은 아니며 단지 규정으로 그렇게 정해진 것입니다. 패치된 mkisofs 프로그램을 쓰면 자동으로 이 부트 카탈로그를 만들어 줍니다만, mkisofs 실행시에 부트 카탈로그를 iso9660 파일시스템의 어디에 넣을 것인지를 지정해주어야 합니다. 보통은 부트이미지와 같은 곳에 boot.catalog 이라는 이름으로 넣어두면 좋습니다.

이제 우리는 boot.img 이라는 파일속에 부트이미지를 담았고, 이제 이것을 iso9660 파일시스템의 루트 디렉토리 밑의 boot/ 디렉토리에 넣을 것입니다. boot.catalog 이라는 이름으로 부트 카탈로그도 같은 디렉토리에 넣겠습니다. bootcd.iso 이라는 화일속에 iso9660 파일시스템을 만드는 명령은 다음과 같습니다:

```
mkisofs -r -b boot/boot.img -c boot/boot.catalog -o bootcd.iso .
```

옵션 -b 는 원하는 부트이미지의 이름이고(패스가 iso9660 디스크의 루트를 기준한 것임에 주목하세요), 옵션 -c 는 부트 카탈로그 파일을 지정한 것입니다. 옵션 -r 은 적절한 화일 소유권과 모드를 지정한 것입니다(mkisofs 맨 페이지를 보세요). 마지막의 "." 은 현재의 디렉토리에 소스들이 있다는 뜻입니다.

이제 보통의 cdrecord 명령으로 CD 를 구워 부팅하면 됩니다.

10.4. 부팅가능한 Win9x 시디롬 만들기

해야 할 일은 원본 시디에 사용된 부팅 이미지를 뽑아내는 것입니다. 하지만 단순히 리눅스 상에서 CD 를 마운트해 dd 로 앞부분의 1440k 를 뽑아 플로피디스크로 복사하거나 `boot.img` 같은 화일로 만들 수가 없습니다. 우선 소스 시디롬으로 부트합시다.

일단 Win98 CD 로 부팅했다면 당신은 A: 이라는 프롬프트를 보게될텐데 이것은 사실 램디스크입니다. 그리고 D: 나 Z: 등은 모든 인스톨매체들이 됩니다. 도스의 diskcopy 명령어를 써서 A: 이미지를 실제의 플로피 드라이브인 B: 로 복사합니다. 명령어는 아래와 같습니다.

```
diskcopy A: B:
```

이것은 dd 와 같은 역할을 합니다. 이렇게 제작된 디스크로부터 부팅한 후, 부팅 과정이 소스 시디에서와 비슷한지를 테스트해보십시요. 이상이 없다면 이 플로피에 dd 명령을 써서 boot.img 같은 파일로 만드세요. 그 다음 과정은 위에서 설명한 것과 동일합니다.

11. 자주 받는 질문들(FAQ : Frequently Asked Question)

질문 **boot/root** 디스크로 부트했는데 아무 일도 생기지 않습니다**.** 어떻게 해야 하나요**?**

답변 위의 7절 부분을 보십시오.

질문 슬랙웨어/데비안/레드햇의 부트디스크들은 어떻게 동작하는 것인가요**?**

답변 위의 9절 부분을 보십시오.

질문 **1440 KB** 를 초과하는 고밀도 디스켓을 사용하려면**?** 자신의 디스켓 드라이브가 지원하는 디스켓 밀도를 확인하려면**?**

답변 다음은 fdutils 프로그램의 작성자인 알라인 크나프(Alain Knaff) 씨의 간략한 답변입니다.

> 이것은 디스켓의 물리적 포맷의 문제라기보다는 오히려 BIOS 에 관련된 문제라 할수 있습니다. 만약 BIOS 가 18 을 초과하는 섹터넘버를 에러(bad)로 판단한다면 더이상 우리가 할수 있는 일은 별로 없습니다. BIOS 를 디스어셈블링한 자료가 부족하기 때문에 이는 시행착오를 거쳐 확인할 수 밖에 없습니다. 하지만, 만일 BIOS 가 ED 디스크(extra density: 36 sectors/track, 2.88MB)를 지원하는 경우에는, 1722K 디스크 역시 지원할 것입니다.
>
> 트랙당 섹터수가 21 섹터를 초과한, 소위 수퍼포멧된 디스크로는 부팅이 안되기 쉽습니다: 사실, 이러한 디스크들은 비표준 사이즈(예를 들면, 섹터당 512 바이트 대신 1024 바이트를 할당)의 섹터를 사용하기 때문에 부팅에 실패하기 쉽습니다. 하지만 이를 위해 특별한 부트섹터 프로그램을 작성한다면 문제를 해결할 수 있습니다. 제 기억이 맞다면, DOS 2M 유틸리티에 이런 기능이 있고, OS/2 의 XDF 유틸리티에도 이런 기능이 있습니다.
>
> 일부 BIOS 들은 18 을 초과하는 섹터넘버는 무조건 에러로 간주해 버립니다. 1722 K 디스크는 21 섹터까지 사용하기 때문에 이 경우 부팅이 안될 것입니다. 이를 확인하는 가장 확실한 테스트 방법은 DOS 나 syslinux 디스켓을 1722 K 로 포맷한 후 부팅가능하게 만들어, 이것으로 부팅해 보는 것입니다. 만일 LILO 를 사용한다면 linear 옵션을 주어서는 안됩니다(linear 옵션을 주면 LILO 는 그 디스켓을 표준 18 sectors/track 디스켓으로 간주하기 때문에 BIOS 가 고밀도 디스켓을 지원함에도 불구하고 부팅에 실패할 것입니다).

질문 부팅가능한 시디롬을 만들려면**?**

답변 10절 절을 보십시요.

질문 부팅가능한 **LS-120** 디스크를 만들려면**?**

답변 필자에게는 LS-120 드라이브가 없기 때문에, 다음 정보는 데이브 시네게(Dave Cinege) 씨가 리눅스 라우터 프로젝트에 제공한 내용 (http://www.linuxrouter.org/floppy.shtml)을 요약한 것입니다.

LS-120 은 IDE 플로피 드라이브의 일종입니다. 표준 3.5 인치 디스크와 호환되며 새로운 120 MB 디스크를 사용할 수 있습니다. 리눅스 v2.0.31 에서는 완벽하게 지원됩니다. 이것으로 부팅하려면 LS-120 을 드라이브 0 으로 특별하게 다루어주는 BIOS 가 반드시 있어야 합니다(이에반해, IDE 디바이스들은 보통 80 에서 시작합니다). BIOS 가 이를 지원하지 않는 경우, Promise Technologies 사의 소형 IDE FloppyMax 카드를 구입하면 이 문제를 해결할 수 있습니다.

커널 부트로더는 LS-120 을 좋아하지 않기 때문에 즉각 멈춰버리게 됩니다. 2m 디스크들도 LS-120 에서는 부트되지 않을 것입니다. 1.44 MB 에서 1.74 MB 디스크들은 깨끗하게 부트될 것입니다. SYSLINUX v1.32 는 120 MB 디스크와 잘 작동합니다. MS-DOS 호환성이 필요없다면 SYSLINX 를 쓰는 대신, 디스크를 파티션 나누어 ext2 나 minix 파일시스템을 쓰는 것이 더 좋을 것입니다.

LILO 는 120 MB 디스크와 잘 작동합니다. 다음은 lilo.conf 의 한 예입니다.

```
boot=/dev/hda
compact
disk=/dev/hda bios=0
install=/floppy/boot.b
map=/floppy/map
image=/floppy/linux
label=Linux
append="load_ramdisk=1"
initrd=/floppy/root.bin
ramdisk=8192
```

"disk=/dev/hda bios=0" 부분이 LS-120 로 부팅하는 요령입니다.

질문 **XYZ** 드라이버를 포함한 부트디스크를 만들려면**?**

답변 가장 쉬운 방법은 가까운 슬랙웨어 미러 사이트에서 슬랙웨어 커널을 받는 것입니다. 슬랙웨어 커널들은 가능한 한 많은 디바이스 드라이버들을 포함하는 포괄적인 커널들이므로 만일 SCSI 나 IDE 컨트롤러를 가지고 있다면 해당 드라이버가 슬랙웨어 커널에 있을 가능성이 높습니다.

`a1` 디렉토리에 가서 당신의 컨트롤러 타입에 맞는 IDE 나 SCSI 커널을 선택하십시오. 선택하신 커널에 대한 xxxxkern.cfg 파일을 보면 해당 드라이버가 그 커널에 들어있는지 확인할 수 있습니다. 원하는 디바이스가 리스트 안에 있다면, 그 커널은 당신 컴퓨터를 부팅시킬 수 있을 것입니다. xxxxkern.tgz 파일을 다운받은 후 본 문서의 부트디스크 제작에 관한 부분에 적힌 방법대로 부트디스크로 복사하십시오.

그 다음, **rdev zImage** 명령을 써서 커널의 루트 디바이스를 확인합니다. 만일 이게 당신이 원하는 루트 디바이스가 아니라면, **rdev** 명령을 써서 루트 디바이스를 바꿔주어야 합니다. 예를들면, 지금 다운받은 커널에는 `/dev/sda2` 가 루트 디바이스로 지정되어 있는데 정작 자신의 루트 디바이스는 `/dev/sda8` 이라는 SCSI 파티션일 수 있는 것입니다. 만일 루트디스켓을 이용하겠다면 **rdev zImage /dev/fd0** 명령으로 루트 파일시스템의 위치가 플로피디스켓임을 알려주어야 합니다.

슬랙웨어 루트디스크의 셋업방법까지 알고 싶어하실지 모르겠는데, 그것은 본 HOWTO 문서의 범위를 벗어나는 내용입니다. 원하시는 분들은 리눅스 설치가이드를 보시거나 슬랙웨어 배포판을 구해보시기 바랍니다. 본 문서의 "참고자료" 부분을 참고하세요.

질문 루트디스켓의 파일들을 새로운 파일들로 갱신하려면 어떻게 해야 하나요**?**

답변 가장 쉬운 방법은 루트디스크의 파일 시스템을 당신이 사용했던 DEVICE (4.2절에서 나왔었습니다)에 역으로 복사해온 후, 그 파일 시스템을 마운트해서 필요한 부분을 갱신합니다. 당신의 루트 파일시스템이 어디서부터 시작하고 얼마만큼의 블록을 차지하는지를 반드시 기억해두어야 합니다.

```
dd if=/dev/fd0 bs=1k skip=ROOTBEGIN count=BLOCKS | gunzip > DEVICE
mount -t ext2 DEVICE /mnt
```

갱신한 다음에는 4.7절 부분에서 했던 것처럼 작업한 후, 루트 파일시스템을 다시 디스크로 옮깁니다. 이때, 새로운 루트 파일시스템의 시작위치를 바꾸지 않았다면 커널을 다시 넣는다거나 램디스크 워드를 재계산해줄 필요는 없습니다.

질문 다시 도스를 쓰고 싶어서 **LILO** 를 제거할까 하는데 어떻게 해야 하나요**?**

답변 사실 이 질문은 부트디스크에 관한 것은 아닙니다만 빈번히 되풀이되는 질문입니다. 리눅스로 하려면 다음 명령을 쓰십시오.

```
/sbin/lilo -u
```

다른 방법으로는 LILO 로 저장해둔 백업을 **dd** 명령을 써서 부트섹터로 복사할 수도 있습니다. 이 방법을 쓰시려면 LILO 에 관한 문서를 참고하시기 바랍니다.

DOS 나 윈도우즈 내에서는 다음 DOS 명령어를 사용하십시오.

```
FDISK /MBR
```

MBR 은 마스터 부트 레코드(Master Boot Record)를 뜻하는 용어입니다. 위의 명령은 파티션 테이블은 건드리지 않으면서 부트섹터를 깨끗한 DOS 부트섹터로 바꾸어 줍니다. 완벽하게 깨끗이해야 한다고 주장하는 사람들은 이 방법에 동의하지 않고있지만 심지어는 LILO 를 제작한 Werner Almesberger 씨 조차도 이 방법을 추천하고 있습니다. 이 방법이 쉽고 확실합니다.

질문 커널과 부트디스크를 둘다 잃어버렸는데 부팅시킬 방법이 있을까요**?**

답변 준비해놓은 부트디스크가 없으시다면, 가장 쉬운 방법은 당신의 디스크 컨트롤러 타입(IDE 혹은 SCSI)에 맞는 슬랙웨어 커널을 구하는 것입니다. 이것은 위의 “ XYZ 드라이버를

포함한 부트디스크를 만들려면? ” 에 대한 답변에서 설명했습니다. 그렇게 구한 커널을 써서 컴퓨터를 부트시킨 후 손상된 부분을 고치시기 바랍니다.

구하신 커널의 루트 디바이스 세팅이 원하시는 디스크 타입과 파티션으로 설정되어있지 않았을 수 있습니다. 예를 들어 슬랙웨어에서는 일반적으로 커널의 SCSI 루트 디바이스가 /dev/sda2 로 잡혀있는데 필자의 리눅스의 루트파티션은 /dev/sda8 로 되어있다고 합시다. 이 경우 커널내의 루트 디바이스 설정을 수정해주어야 합니다.

심지어는 가진 것이 달랑 커널 하나와 DOS 혹은 그 외 다른 운영체제뿐이더라도 그걸로도 커널내의 루트 디바이스와 램디스크 설정을 바꿀수 있습니다.

rdev 명령은 커널파일내에 고정되어있는 오프셋의 값을 바꿔줌으로써 커널의 세팅을 바꾸는 것입니다. 따라서 현재 어떤 시스템을 갖고 있든지 간에 hex 에디터만 쓸 수 있다면 같은 작업을 해낼 수 있습니다. -- 한 예로 DOS 의 노턴 유틸리티 디스크에디터를 사용할 수도 있습니다. 에디터를 써서 커널내의 다음 오프셋값들을 체크하고 필요하다면 이를 수정하시면 됩니다.

```
HEX     DEC  DESCRIPTION
0x01F8  504  Low byte of RAMDISK word
0x01F9  505  High byte of RAMDISK word
0x01FC  508  Root minor device number - see below
0X01FD  509  Root major device number - see below
```

램디스크 워드의 해석은 위의 6.3절 부분에 나와 있습니다.

메이저, 마이너 디바이스 넘버들은 루트 파일시스템이 마운트될 디바이스로 설정되어야 합니다. 쓸만한 값들은 다음과 같습니다.

```
DEVICE          MAJOR MINOR
/dev/fd0            2     0   1st floppy drive
/dev/hda1           3     1   partition 1 on 1st IDE drive
/dev/sda1           8     1   partition 1 on 1st SCSI drive
/dev/sda8           8     8   partition 8 on 1st SCSI drive
```

일단 이 값들을 설정했다면 이제 노턴 유틸리티 디스크 에디터나 **rawrite.exe** 같은 프로그램을 써서 파일을 디스켓 상에 써넣어야 합니다. **rawrite.exe** 프로그램은 모든 배포본에 다 포함되어 있습니다. 이것은 도스프로그램으로서 어떤 파일을 파일시스템 내에 쓰는 것이 아니라 부트섹터부터 그대로(raw 하게) 디스크로 써넣어 버리는 프로그램입니다. 노턴 유틸리티를 사용하신다면 파일을 디스크의 물리적인 첫부분에서부터 그대로 써넣어야만 합니다.

질문 **boot/root** 디스켓의 복사본을 만들려면**?**

답변 자성 매체는 시간이 지남에 따라 그 자기적 특성이 저하되기 때문에, 원본을 읽을 수 없게 될 경우에 대비해 복구용 디스켓을 여벌로 가지고 있는 것이 좋습니다.

부팅 가능한 디스켓이든 유틸리티 디스켓이든간에 어떤 디스켓의 복사본을 만드는 가장 손쉬운 방법은 **dd** 명령으로 원본디스켓의 내용을 하드디스크 상의 파일로 복사해 온 후, 같은 명령으로 그 파일을 새로운 디스켓에 복사해 넣는 것입니다. 이때 디스켓을 마운트할 필요가 없으며 또 마운트해서도 안됩니다. 왜냐하면 **dd** 명령은 raw device 인터페이스를 사용하기 때문입니다.

원본을 복사하기 위해서는 다음과 같은 명령을 씁니다.

```
dd if=DEVICENAME of=FILENAME
```

여기서 DEVICENAME 은 디스켓 드라이브의 디바이스의 이름이고 FILENAME 은 하드디스크에 생성되는 파일의 이름입니다. **dd** 명령에서 `count` 파라메터를 생략하면 디스크 전체(고밀도 디스켓의 경우 2880 블록)를 복사하게 됩니다.

얻어낸 파일을 새로운 디스켓으로 복사하려면 새 디스켓을 넣고 명령을 반대로 내립니다.

```
dd if=FILENAME of=DEVICENAME
```

이상은 당신이 오직 하나의 디스크 드라이브를 가지고 있다고 가정한 것입니다. 만일 같은 타입의 디스크 드라이브를 두 개 가지고 있다면, 다음 명령어로 디스켓을 복사할 수 있습니다.

```
dd if=/dev/fd0 of=/dev/fd1
```

질문 매번 부트할 때마다 "**ahaxxxx=nn,nn,nn**" 식으로 입력하지 않고 부트할 수는 없나요**?**

답변 디스크 디바이스가 자동으로 감지되지 않는다면 다음처럼 커널에 디바이스 파라메터 문자열을 써 주어야 합니다. 예를 들면 다음과 같은 식입니다.

```
aha152x=0x340,11,3,1
```

LILO 사용시 이 파라메터 문자열을 전달하는 몇 가지 방법이 있습니다.

- 시스템이 LILO 를 통해 부트될 때마다 명령어 라인에 문자열을 써주는 방법. 하지만 이것은 귀찮은 방법입니다.
- LILO 의 `lock` 키워드를 써서 원하는 명령어 라인을 디폴트로 저장시키는 방법. 이렇게 하면 LILO 는 부트할 때마다 매번 이 옵션을 사용하게 됩니다.
- LILO 설정파일내에 `append=` 구문을 쓰는 방법. 이때 파라메터 문자열은 반드시 인용부호로 감싸주어야만 합니다.

다음은 위의 파라메터 문자열을 사용한 명령어 라인의 예입니다.

```
zImage  aha152x=0x340,11,3,1 root=/dev/sda1 lock
```

이것은 디바이스 파라메터 문자열을 넘겨주면서 동시에 커널로 하여금 루트 디바이스를 `/dev/sda1` 로 설정케 한 후, 명령어 라인 전체를 저장시켜 차후로 부트할 때마다 이를 사용합니다.

다음은 APPEND 구문의 예입니다.

```
APPEND = "aha152x=0x340,11,3,1"
```

파라메터 문자열은 명령어 라인에서는 절대로 인용부호를 써서는 안되며, 반대로 `APPEND` 구문에서는 반드시 인용부호를 같이 써야 합니다.

또, 파라메터 문자열대로 제대로 동작하게 하려면 커널은 반드시 그 디스크 타입에 해당하는 드라이버를 가지고 있어야만 합니다. 만일 해당 드라이버를 가지고 있지 않다면 그 파라메터 문자열은 아무런 작용을 못하는 있으나마나한 존재가 되므로 이럴때는 필요한 드라이버를 포함하는 커널을 다시 만들어 설치해야 합니다. 커널을 제작시의 자세한 사항은 `/usr/src/linux` 로 가셔서 README 파일을 읽으시고 리눅스 FAQ 와 리눅스 설치 HOWTO 를 읽어 보십시오. 아니면 그 디스크 타입에 맞는 일반적인 커널을 구해 설치하셔도 됩니다.

LILO 설치를 시험해 보시기 전에 반드시 LILO 문서를 읽어보시기 바랍니다. BOOT 구문을 부주의하게 사용하면 파티션이 손상될 수 있습니다.

질문 부트할 때 "**`A: cannot execute B`**" 라는 에러가 났습니다**.** 왜인가요**?**

답변 어떤 프로그램의 이름이 다른 유틸리티 프로그램의 코드내에서 직접 코딩(hardcoded)된 경우가 있습니다. 이런 경우가 어디에나 있는 것은 아닙니다. 하지만 이러한 경우는 왜 어떤 실행파일이 분명히 존재하는데도 불구하고 특정 프로그램이 그 파일을 찾지 못하는가를 설명해줍니다. 특정 프로그램이 코드내에서 다른 파일의 이름을 사용하고 있는지 여부를 확인하려면 **strings** 명령을 쓴 후 그 출력을 **grep** 으로 파이프 받아 확인해보면 됩니다.

이런 경우의 몇가지 실제 예가 알려져 있습니다.

- 어떤 리눅스 버전에서의 **shutdown** 프로그램은 그 코드내에 `/etc/reboot` 라는 이름을 바로 사용하고 있습니다. 따라서 이 경우 `reboot` 프로그램은 반드시 `/etc` 디렉토리 밑에 위치해야만 합니다.
- 커널이 **init** 를 찾지 못해서 문제가 되는 경우도 있었습니다.

이런 문제들을 해결하기 위해서는, 해당 프로그램을 올바른 디렉토리에 두거나 설정파일(예를 들면 `inittab`)들을 고쳐 올바른 디렉토리를 가리키도록 해야 합니다. 아무래도 자신없다면 하드디스크의 환경과 동일한 환경을 만들어 주십시오. 그다음, 프로그램들을 하드디스크에서와 동일한 디렉토리에 두고, 하드디스크에서 쓰는 `inittab`, `/etc/rc.d` 와 동일한 파일을 사용해 봅니다.

질문 제 커널은 램디스크를 지원합니다만 램디스크를 **0 K** 로 초기화시켜 버립니다**.**

답변 이런 일이 발생하면 부팅시에 다음과 같은 커널 메시지가 뜹니다.

```
Ramdisk driver initialized : 16 ramdisks of 0K size
```

이는 틀림없이 부트시에 커널 파라메터가 램디스크의 크기를 0 으로 세팅했기 때문일 겁니다. 아마도 LILO 설정파일의 파라메터가 다음과 같이 설정되어있는 것을 간과하셨을 것입니다.

```
ramdisk= 0
```

일부 오래된 배포판에 포함된 LILO 설정의 샘플 파일에 이런 옵션이 들어 있는 수가 있습니다. 지금 이 샘플 설정파일이 커널을 세팅하고있는 것입니다. 위와 같은 라인이 있다면 삭제하세요.

만일 0 K 로 설정되어있는 램디스크를 사용하려 시도한다면 그 결과는 예상할 수 없습니다. 이는 커널 패닉으로 이어질 수도 있습니다.

A. 참고자료

패키지를 가져올 때는 특별한 이유가 없다면 항상 최신 버전을 구하십시오.

A.1. 미리 만들어져 있는 부트디스크

다음은 배포판의 부트디스크를 구할 수 있는 곳입니다. 될수 있으면 미러 사이트를 이용하셔서 한곳에만 부하가 집중되지 않도록 해줍시다.

- Slackware bootdisks (http://metalab.unc.edu/pub/Linux/distributions/slackware/bootdsks.144/), rootdisks (http://metalab.unc.edu/pub/Linux/distributions/slackware/rootdsks/) and Slackware mirror sites (http://www.slackware.com/getslack/)
- RedHat bootdisks (http://metalab.unc.edu/pub/Linux/distributions/redhat/current/i386/images/) and Red Hat mirror sites (http://www.redhat.com/mirrors.html)
- Debian bootdisks (ftp://ftp.debian.org/pub/debian/dists/stable/main/disks-i386/current/) and Debian mirror sites (ftp://ftp.debian.org/pub/debian/README.mirrors.html)

배포본 부트디스크 외에도 다음의 복구 디스크 이미지들을 쓸 수도 있습니다. 특별히 구하는 곳이 명시되지 않은 것들은 다음 디렉토리에서 구할 수 있습니다.
`http://metalab.unc.edu/pub/Linux/system/recovery/!INDEX.html`

- Tom Oehser 씨가 제작한 `tomsrtbt` 는 커널 2.0 기반의 한장짜리 boot/root 디스크입니다. 많은 프로그램들을 지원하며 기능이 많습니다. IDE, SCSI, 테이프, 네트웍 어댑터, PCMCIA 기타 여러가지를 지원합니다. 디스크 복구 및 수리에 관련된 약 100 개 이상의 유틸리티와 툴이 포함되어 있습니다. 이미지를 풀어 재구축할수 있는 스크립트들이 포함되어 있으므로 필요하다면 새로운 것을 첨가할 수 있습니다.
- John Comyns 씨가 제작한 `rescue02` 는 커널 1.3.84 기반의 복구 디스크입니다. IDE, 아답텍 1542, NCR53C7,8xx 를 지원합니다. ELF 바이너리를 사용하지만 충분한 명령어들을 가지고 있으므로 어떤 시스템에도 사용할 수 있습니다. 부팅 후에는 그밖의 SCSI 카드들에 대한 모듈도 로드할 수 있습니다. 이 디스크는 3 메가의 램디스크를 사용하므로 4 mb 의 램을 가진 시스템에서는 쓸 수 없을 것입니다.
- Sergei Viznyuk 씨가 제작한 `resque_disk-2.0.22` 는 다기능의 boot/root 디스크로서 커널 2.0.22 를 기반으로 합니다. IDE, 각종 SCSI 컨트롤러들, ELF/A.OUT 에 대한 지원을 커널에 내장하고 있습니다. 또한 많은 모듈들과 하드디스크의 수리와 복구에 유용한 많은 유틸리티들을 가지고 있습니다.

- `cramdisk` 이미지들 (http://metalab.unc.edu/pub/Linux/system/recovery/images)은 커널 2.0.23을 기반으로 제작되었으며, 4 MB 나 8 MB 머신에서 사용가능합니다. math emulation, 네트워킹(PPP 와 다이얼링 스크립트, NE2000, 3C509), 병렬포트 ZIP 드라이브 등에 대한 지원을 포함하고 있습니다. 이 디스켓 이미지들은 4 MB 이상의 램을 가진 386 에서 사용가능합니다. MSDOS 지원이 포함되어 있으므로 도스파티션으로 다운받을 수 있습니다.

A.2. 복구 패키지들

metalab.unc.edu 에는 복구디스크를 만드는 패키지가 몇 가지 있습니다. 이 패키지들을 쓸 경우, 당신이 포함시키고자하는 파일들을 지정해 주기만 하면 소프트웨어가 어느정도 자동적으로 부트디스크를 만들어 나갑니다. 더 자세한 정보는 `http://metalab.unc.edu/pub/Linux/system/recovery/!INDEX.html` 을 참고하십시오. 파일의 날짜를 주의 깊게 체크하십시오**.** 패키지들 중 몇몇은 지난 수년간 갱신되지 않았기 때문에 램디스크로 로딩되는 압축 루트 파일시스템을 만들수 생성하지 못합니다. 필자들이 아는 바로는 오직 Yard (http://www.croftj.net/~fawcett/yard/index.html) 만이 이를 지원합니다.

A.3. LILO -- the Linux loader

Werner Almesberger 씨가 작성한 탁월한 부트로더입니다. LILO 의 설명문서에는 부트섹터의 내용과 부트 프로세스의 초기단계에 관한 정보가 실려 있습니다.

ftp 인 ftp://tsx-11.mit.edu/pub/linux/packages/lilo/ 에서 구할수 있습니다. Metalab 과 그 미러사이트에서도 구하실 수 있습니다.

A.4. 램디스크 사용법

램디스크 코드의 동작원리에 대한 탁월한 설명은 리눅스 커널에 따라오는 문서에서 찾으실 수 있습니다. `/usr/src/linux/Documentation/ramdisk.txt` 를 보십시오. Paul Gortmaker 씨가 썼으며 압축된 램디스크 생성에 관한 부분이 포함되어 있습니다.

A.5. 리눅스의 부트 과정

리눅스의 부트 과정에 관한 보다 자세한 사항은 다음을 참고하십시오.

- *리눅스 시스템 관리자 가이드* (http://linuxdoc.org/LDP/sag/c1596.html) 에 부팅에 관한 절이 있습니다.

- LILO “Technical overview” (http://metalab.unc.edu/pub/Linux/system/boot/lilo/lilo-t-21.ps.gz) 에는 부트프로세스에서부터 커널이 시작되는 단계까지에 관한 기술적인 사항 및 저수준에서의 동작에 관한 확실한 설명이 있습니다.
- 소스코드야 말로 궁극적인 안내서입니다. 아래의 파일들은 부트프로세스와 관련된 몇가지 커널 파일들입니다. 리눅스 커널 소스 코드를 가지고 계신다면 `/usr/src/linux` 아래에 있을 것입니다. 다른 방법으로는, Shigio Yamaguchi (shigio@tamacom.com) 씨가 아주 훌륭한 하이퍼텍스트 커널 브라우저 (http://www.tamacom.com/tour/linux/index.html)를 제공하고 있습니다. 다음은 관련있는 파일들입니다.

  `arch/i386/boot/bootsect.S` 과 `setup.S`

  부트섹터 자체에 해당되는 어셈블리 코드가 들어있습니다.

  `arch/i386/boot/compressed/misc.c`

  커널의 압축을 해제하는 코드가 들어있습니다.

  `arch/i386/kernel/`

  커널을 초기화시키는 코드가 이 디렉토리에 들어있습니다. `setup.c` 은 램디스크 워드를 정의하고 있습니다.

  `drivers/block/rd.c`

  램디스크 드라이버가 들어있습니다. **rd_load** 와 **rd_load_image** 프로시저가 디바이스에서 램디스크로 블록들을 로드합니다. **identify_ramdisk_image** 프로시저는 어떤 종류의 파일시스템이 발견되었는지, 또 그것이 압축된 상태인지 아닌지를 판별합니다.

B. LILO 부트에러 코드

다음 에러들에 관한 질문이 유즈넷 상에서 빈번한지라 필자는 공공 서비스 차원에서 문서에 이를 포함시켰습니다. 다음은 Werner Almsberger 씨의 LILO 사용자 안내서 (http://metalab.unc.edu/pub/Linux/system/boot/lilo/lilo-u-21.ps.gz). 에서 인용했습니다.

LILO 가 스스로를 로드할 때는 `LILO` 라는 단어가 디스플레이 됩니다. 각 문자는 어떤 특정한 작업이 수행되기 직전이나 직후에 출력됩니다. 만일 LILO 가 어느 단계에서 실패하면 그때까지 출력된 문자는 문제가 무엇인지를 나타내는 역할을 합니다.

Output	Problem
(아무 글자도 나타나지 않을 때)	LILO 는 전혀 로드되지 않은 것입니다. LILO 가 아예 설치되지 않았거나 부트섹터가 위치하는 파티션이 active 하지 않기 때문입니다.

Output	Problem
L	부트로더의 첫 단계는 로드되어 시작되었지만 부트로더의 두번째 단계가 로드되지 못한 것입니다. 두자리 수의 에러코드들은 어떤 타입의 문제가 발생했는지를 나타냅니다("디스크 에러 코드" 부분을 참고하세요). 이 상태는 보통 매체에 이상이 있거나 지오메트리의 불일치인 경우입니다(예를 들면 디스크 파라메터를 잘못 준 경우).
LI	부트로더의 첫 단계가 부트로더의 두번째 단계를 로드하기는 했는데 그것을 실행시키는 데 실패한 것입니다. 이것은 지오메트리의 불일치(geometry mismatch)이거나 map installer 를 실행시키지 않은 채 `/boot/boot.b` 를 옮긴데서 기인합니다.
LIL	부트로더의 두번째 단계가 시작되었지만 이것이 map 파일에서 디스크립터 테이블을 로드하지 못한 것입니다. 이런 증상은 대부분 매체의 이상이거나 지오메트리가 일치하지 않기 때문입니다.
LIL?	부트로더의 두번째 단계가 잘못된 어드레스로 로드된 것입니다. 이런 증상은 사소한 지오메트리 불일치이거나 map installer 를 구동시키지 않은 채로 `/boot/boot.b` 를 이동시켰을 때의 전형적인 현상입니다.
LIL-	디스크립터 테이블이 잘못된 경우입니다. 이는 지오메트리 불일치이거나 map installer를 구동시키지 않은 채로 `boot/boot.b` 를 이동시켰기 때문입니다.
LILO	LILO 의 모든 부분이 성공적으로 로드된 것입니다.

LILO 가 부트 이미지를 로드할 때 BIOS 가 에러를 낸다면 각각의 에러메시지가 디스플레이 됩니다. 이 코드들은 `0x00` 부터 `0xbb` 까지 입니다. 각 코드의 해석은 LILO 사용자 안내서를 보십시오.

C. 루트 파일시스템 견본

```
/:
drwx--x--x   2 root     root         1024 Nov  1 15:39 bin
drwx--x--x   2 root     root         4096 Nov  1 15:39 dev
drwx--x--x   3 root     root         1024 Nov  1 15:39 etc
drwx--x--x   4 root     root         1024 Nov  1 15:39 lib
drwx--x--x   5 root     root         1024 Nov  1 15:39 mnt
drwx--x--x   2 root     root         1024 Nov  1 15:39 proc
drwx--x--x   2 root     root         1024 Nov  1 15:39 root
drwx--x--x   2 root     root         1024 Nov  1 15:39 sbin
drwx--x--x   2 root     root         1024 Nov  1 15:39 tmp
drwx--x--x   7 root     root         1024 Nov  1 15:39 usr
drwx--x--x   5 root     root         1024 Nov  1 15:39 var

/bin:
-rwx--x--x   1 root     root        62660 Nov  1 15:39 ash
-rwx--x--x   1 root     root         9032 Nov  1 15:39 cat
-rwx--x--x   1 root     root        10276 Nov  1 15:39 chmod
-rwx--x--x   1 root     root         9592 Nov  1 15:39 chown
-rwx--x--x   1 root     root        23124 Nov  1 15:39 cp
```

```
-rwx--x--x   1 root     root        23028 Nov  1 15:39 date
-rwx--x--x   1 root     root        14052 Nov  1 15:39 dd
-rwx--x--x   1 root     root        14144 Nov  1 15:39 df
-rwx--x--x   1 root     root        69444 Nov  1 15:39 egrep
-rwx--x--x   1 root     root          395 Nov  1 15:39 false
-rwx--x--x   1 root     root        69444 Nov  1 15:39 fgrep
-rwx--x--x   1 root     root        69444 Nov  1 15:39 grep
-rwx--x--x   3 root     root        45436 Nov  1 15:39 gunzip
-rwx--x--x   3 root     root        45436 Nov  1 15:39 gzip
-rwx--x--x   1 root     root         8008 Nov  1 15:39 hostname
-rwx--x--x   1 root     root        12736 Nov  1 15:39 ln
-rws--x--x   1 root     root        15284 Nov  1 15:39 login
-rwx--x--x   1 root     root        29308 Nov  1 15:39 ls
-rwx--x--x   1 root     root         8268 Nov  1 15:39 mkdir
-rwx--x--x   1 root     root         8920 Nov  1 15:39 mknod
-rwx--x--x   1 root     root        24836 Nov  1 15:39 more
-rws--x--x   1 root     root        37640 Nov  1 15:39 mount
-rwx--x--x   1 root     root        12240 Nov  1 15:39 mt
-rwx--x--x   1 root     root        12932 Nov  1 15:39 mv
-r-x--x--x   1 root     root        12324 Nov  1 15:39 ps
-rwx--x--x   1 root     root         5388 Nov  1 15:39 pwd
-rwx--x--x   1 root     root        10092 Nov  1 15:39 rm
lrwxrwxrwx   1 root     root            3 Nov  1 15:39 sh -> ash
-rwx--x--x   1 root     root        25296 Nov  1 15:39 stty
-rws--x--x   1 root     root        12648 Nov  1 15:39 su
-rwx--x--x   1 root     root         4444 Nov  1 15:39 sync
-rwx--x--x   1 root     root       110668 Nov  1 15:39 tar
-rwx--x--x   1 root     root        19712 Nov  1 15:39 touch
-rwx--x--x   1 root     root          395 Nov  1 15:39 true
-rws--x--x   1 root     root        19084 Nov  1 15:39 umount
-rwx--x--x   1 root     root         5368 Nov  1 15:39 uname
-rwx--x--x   3 root     root        45436 Nov  1 15:39 zcat

/dev:
lrwxrwxrwx   1 root     root            6 Nov  1 15:39 cdrom -> cdu31a
brw-rw-r--   1 root     root      15,   0 May  5  1998 cdu31a
crw-------   1 root     root       4,   0 Nov  1 15:29 console
crw-rw-rw-   1 root     uucp       5,  64 Sep  9 19:46 cua0
crw-rw-rw-   1 root     uucp       5,  65 May  5  1998 cua1
crw-rw-rw-   1 root     uucp       5,  66 May  5  1998 cua2
crw-rw-rw-   1 root     uucp       5,  67 May  5  1998 cua3
brw-rw----   1 root     floppy     2,   0 Aug  8 13:54 fd0
brw-rw----   1 root     floppy     2,  36 Aug  8 13:54 fd0CompaQ
brw-rw----   1 root     floppy     2,  84 Aug  8 13:55 fd0D1040
brw-rw----   1 root     floppy     2,  88 Aug  8 13:55 fd0D1120
brw-rw----   1 root     floppy     2,  12 Aug  8 13:54 fd0D360
brw-rw----   1 root     floppy     2,  16 Aug  8 13:54 fd0D720
brw-rw----   1 root     floppy     2, 120 Aug  8 13:55 fd0D800
brw-rw----   1 root     floppy     2,  32 Aug  8 13:54 fd0E2880
brw-rw----   1 root     floppy     2, 104 Aug  8 13:55 fd0E3200
brw-rw----   1 root     floppy     2, 108 Aug  8 13:55 fd0E3520
brw-rw----   1 root     floppy     2, 112 Aug  8 13:55 fd0E3840
brw-rw----   1 root     floppy     2,  28 Aug  8 13:54 fd0H1440
brw-rw----   1 root     floppy     2, 124 Aug  8 13:55 fd0H1600
brw-rw----   1 root     floppy     2,  44 Aug  8 13:55 fd0H1680
brw-rw----   1 root     floppy     2,  60 Aug  8 13:55 fd0H1722
brw-rw----   1 root     floppy     2,  76 Aug  8 13:55 fd0H1743
brw-rw----   1 root     floppy     2,  96 Aug  8 13:55 fd0H1760
brw-rw----   1 root     floppy     2, 116 Aug  8 13:55 fd0H1840
brw-rw----   1 root     floppy     2, 100 Aug  8 13:55 fd0H1920
lrwxrwxrwx   1 root     root            7 Nov  1 15:39 fd0H360 -> fd0D360
```

```
lrwxrwxrwx   1 root     root            7 Nov  1 15:39 fd0H720 -> fd0D720
brw-rw----   1 root     floppy     2,  52 Aug  8 13:55 fd0H820
brw-rw----   1 root     floppy     2,  68 Aug  8 13:55 fd0H830
brw-rw----   1 root     floppy     2,   4 Aug  8 13:54 fd0d360
brw-rw----   1 root     floppy     2,   8 Aug  8 13:54 fd0h1200
brw-rw----   1 root     floppy     2,  40 Aug  8 13:54 fd0h1440
brw-rw----   1 root     floppy     2,  56 Aug  8 13:55 fd0h1476
brw-rw----   1 root     floppy     2,  72 Aug  8 13:55 fd0h1494
brw-rw----   1 root     floppy     2,  92 Aug  8 13:55 fd0h1600
brw-rw----   1 root     floppy     2,  20 Aug  8 13:54 fd0h360
brw-rw----   1 root     floppy     2,  48 Aug  8 13:55 fd0h410
brw-rw----   1 root     floppy     2,  64 Aug  8 13:55 fd0h420
brw-rw----   1 root     floppy     2,  24 Aug  8 13:54 fd0h720
brw-rw----   1 root     floppy     2,  80 Aug  8 13:55 fd0h880
brw-rw----   1 root     disk       3,   0 May  5  1998 hda
brw-rw----   1 root     disk       3,   1 May  5  1998 hda1
brw-rw----   1 root     disk       3,   2 May  5  1998 hda2
brw-rw----   1 root     disk       3,   3 May  5  1998 hda3
brw-rw----   1 root     disk       3,   4 May  5  1998 hda4
brw-rw----   1 root     disk       3,   5 May  5  1998 hda5
brw-rw----   1 root     disk       3,   6 May  5  1998 hda6
brw-rw----   1 root     disk       3,  64 May  5  1998 hdb
brw-rw----   1 root     disk       3,  65 May  5  1998 hdb1
brw-rw----   1 root     disk       3,  66 May  5  1998 hdb2
brw-rw----   1 root     disk       3,  67 May  5  1998 hdb3
brw-rw----   1 root     disk       3,  68 May  5  1998 hdb4
brw-rw----   1 root     disk       3,  69 May  5  1998 hdb5
brw-rw----   1 root     disk       3,  70 May  5  1998 hdb6
crw-r-----   1 root     kmem       1,   2 May  5  1998 kmem
crw-r-----   1 root     kmem       1,   1 May  5  1998 mem
lrwxrwxrwx   1 root     root           12 Nov  1 15:39 modem -> ttyS1
lrwxrwxrwx   1 root     root           12 Nov  1 15:39 mouse -> psaux
crw-rw-rw-   1 root     root       1,   3 May  5  1998 null
crwxrwxrwx   1 root     root      10,   1 Oct  5 20:22 psaux
brw-r-----   1 root     disk       1,   1 May  5  1998 ram
brw-rw----   1 root     disk       1,   0 May  5  1998 ram0
brw-rw----   1 root     disk       1,   1 May  5  1998 ram1
brw-rw----   1 root     disk       1,   2 May  5  1998 ram2
brw-rw----   1 root     disk       1,   3 May  5  1998 ram3
brw-rw----   1 root     disk       1,   4 May  5  1998 ram4
brw-rw----   1 root     disk       1,   5 May  5  1998 ram5
brw-rw----   1 root     disk       1,   6 May  5  1998 ram6
brw-rw----   1 root     disk       1,   7 May  5  1998 ram7
brw-rw----   1 root     disk       1,   8 May  5  1998 ram8
brw-rw----   1 root     disk       1,   9 May  5  1998 ram9
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 ramdisk -> ram0
***  I have only included devices for the IDE partitions I use.
***  If you use SCSI, then use the /dev/sdXX devices instead.
crw-------   1 root     root       4,   0 May  5  1998 tty0
crw--w----   1 root     tty        4,   1 Nov  1 15:39 tty1
crw-------   1 root     root       4,   2 Nov  1 15:29 tty2
crw-------   1 root     root       4,   3 Nov  1 15:29 tty3
crw-------   1 root     root       4,   4 Nov  1 15:29 tty4
crw-------   1 root     root       4,   5 Nov  1 15:29 tty5
crw-------   1 root     root       4,   6 Nov  1 15:29 tty6
crw-------   1 root     root       4,   7 May  5  1998 tty7
crw-------   1 root     tty        4,   8 May  5  1998 tty8
crw-------   1 root     tty        4,   9 May  8 12:57 tty9
crw-rw-rw-   1 root     root       4,  65 Nov  1 12:17 ttyS1
crw-rw-rw-   1 root     root       1,   5 May  5  1998 zero
```

```
/etc:
-rw-------   1 root     root          164 Nov  1 15:39 conf.modules
-rw-------   1 root     root          668 Nov  1 15:39 fstab
-rw-------   1 root     root           71 Nov  1 15:39 gettydefs
-rw-------   1 root     root          389 Nov  1 15:39 group
-rw-------   1 root     root          413 Nov  1 15:39 inittab
-rw-------   1 root     root           65 Nov  1 15:39 issue
-rw-r--r--   1 root     root          746 Nov  1 15:39 ld.so.cache
-rw-------   1 root     root           32 Nov  1 15:39 motd
-rw-------   1 root     root          949 Nov  1 15:39 nsswitch.conf
drwx--x--x   2 root     root         1024 Nov  1 15:39 pam.d
-rw-------   1 root     root          139 Nov  1 15:39 passwd
-rw-------   1 root     root          516 Nov  1 15:39 profile
-rwx--x--x   1 root     root          387 Nov  1 15:39 rc
-rw-------   1 root     root           55 Nov  1 15:39 shells
-rw-------   1 root     root          774 Nov  1 15:39 termcap
-rw-------   1 root     root           78 Nov  1 15:39 ttytype
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 utmp -> ../var/run/utmp
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 wtmp -> ../var/log/wtmp

/etc/pam.d:
-rw-------   1 root     root          356 Nov  1 15:39 other

/lib:
-rwxr-xr-x   1 root     root        45415 Nov  1 15:39 ld-2.0.7.so
lrwxrwxrwx   1 root     root           11 Nov  1 15:39 ld-linux.so.2 -> ld-2.0.7.so
-rwxr-xr-x   1 root     root       731548 Nov  1 15:39 libc-2.0.7.so
lrwxrwxrwx   1 root     root           13 Nov  1 15:39 libc.so.6 -> libc-2.0.7.so
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libcom_err.so.2 -> libcom_err.so.2.0
-rwxr-xr-x   1 root     root         6209 Nov  1 15:39 libcom_err.so.2.0
-rwxr-xr-x   1 root     root       153881 Nov  1 15:39 libcrypt-2.0.7.so
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libcrypt.so.1 -> libcrypt-2.0.7.so
-rwxr-xr-x   1 root     root        12962 Nov  1 15:39 libdl-2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libdl.so.2 -> libdl-2.0.7.so
lrwxrwxrwx   1 root     root           16 Nov  1 15:39 libext2fs.so.2 -> libext2fs.so.2.4
-rwxr-xr-x   1 root     root        81382 Nov  1 15:39 libext2fs.so.2.4
-rwxr-xr-x   1 root     root        25222 Nov  1 15:39 libnsl-2.0.7.so
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 libnsl.so.1 -> libnsl-2.0.7.so
-rwx--x--x   1 root     root       178336 Nov  1 15:39 libnss_files-2.0.7.so
lrwxrwxrwx   1 root     root           21 Nov  1 15:39 libnss_files.so.1 -> libnss_files-
2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libpam.so.0 -> libpam.so.0.64
-rwxr-xr-x   1 root     root        26906 Nov  1 15:39 libpam.so.0.64
lrwxrwxrwx   1 root     root           19 Nov  1 15:39 libpam_misc.so.0 -
> libpam_misc.so.0.64
-rwxr-xr-x   1 root     root         7086 Nov  1 15:39 libpam_misc.so.0.64
-r-xr-xr-x   1 root     root        35615 Nov  1 15:39 libproc.so.1.2.6
lrwxrwxrwx   1 root     root           15 Nov  1 15:39 libpwdb.so.0 -> libpwdb.so.0.54
-rw-r--r--   1 root     root       121899 Nov  1 15:39 libpwdb.so.0.54
lrwxrwxrwx   1 root     root           19 Nov  1 15:39 libtermcap.so.2 -> libtermcap.so.2.0.8
-rwxr-xr-x   1 root     root        12041 Nov  1 15:39 libtermcap.so.2.0.8
-rwxr-xr-x   1 root     root        12874 Nov  1 15:39 libutil-2.0.7.so
lrwxrwxrwx   1 root     root           16 Nov  1 15:39 libutil.so.1 -> libutil-2.0.7.so
lrwxrwxrwx   1 root     root           14 Nov  1 15:39 libuuid.so.1 -> libuuid.so.1.1
-rwxr-xr-x   1 root     root         8039 Nov  1 15:39 libuuid.so.1.1
drwx--x--x   3 root     root         1024 Nov  1 15:39 modules
drwx--x--x   2 root     root         1024 Nov  1 15:39 security

/lib/modules:
drwx--x--x   4 root     root         1024 Nov  1 15:39 2.0.35
```

```
/lib/modules/2.0.35:
drwx--x--x   2 root     root         1024 Nov  1 15:39 block
drwx--x--x   2 root     root         1024 Nov  1 15:39 cdrom

/lib/modules/2.0.35/block:
drwx------   1 root     root         7156 Nov  1 15:39 loop.o

/lib/modules/2.0.35/cdrom:
drwx------   1 root     root        24108 Nov  1 15:39 cdu31a.o

/lib/security:
-rwx--x--x   1 root     root         8771 Nov  1 15:39 pam_permit.so

***  Directory stubs for mounting
/mnt:
drwx--x--x   2 root     root         1024 Nov  1 15:39 cdrom
drwx--x--x   2 root     root         1024 Nov  1 15:39 floppy

/proc:

/root:
-rw-------   1 root     root          176 Nov  1 15:39 .bashrc
-rw-------   1 root     root          182 Nov  1 15:39 .cshrc
-rwx--x--x   1 root     root          455 Nov  1 15:39 .profile
-rw-------   1 root     root         4014 Nov  1 15:39 .tcshrc

/sbin:
-rwx--x--x   1 root     root        23976 Nov  1 15:39 depmod
-rwx--x--x   2 root     root       274600 Nov  1 15:39 e2fsck
-rwx--x--x   1 root     root        41268 Nov  1 15:39 fdisk
-rwx--x--x   1 root     root         9396 Nov  1 15:39 fsck
-rwx--x--x   2 root     root       274600 Nov  1 15:39 fsck.ext2
-rwx--x--x   1 root     root        29556 Nov  1 15:39 getty
-rwx--x--x   1 root     root         6620 Nov  1 15:39 halt
-rwx--x--x   1 root     root        23116 Nov  1 15:39 init
-rwx--x--x   1 root     root        25612 Nov  1 15:39 insmod
-rwx--x--x   1 root     root        10368 Nov  1 15:39 kerneld
-rwx--x--x   1 root     root       110400 Nov  1 15:39 ldconfig
-rwx--x--x   1 root     root         6108 Nov  1 15:39 lsmod
-rwx--x--x   2 root     root        17400 Nov  1 15:39 mke2fs
-rwx--x--x   1 root     root         4072 Nov  1 15:39 mkfs
-rwx--x--x   2 root     root        17400 Nov  1 15:39 mkfs.ext2
-rwx--x--x   1 root     root         5664 Nov  1 15:39 mkswap
-rwx--x--x   1 root     root        22032 Nov  1 15:39 modprobe
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 reboot -> halt
-rwx--x--x   1 root     root         7492 Nov  1 15:39 rmmod
-rwx--x--x   1 root     root        12932 Nov  1 15:39 shutdown
lrwxrwxrwx   1 root     root            6 Nov  1 15:39 swapoff -> swapon
-rwx--x--x   1 root     root         5124 Nov  1 15:39 swapon
lrwxrwxrwx   1 root     root            4 Nov  1 15:39 telinit -> init
-rwx--x--x   1 root     root         6944 Nov  1 15:39 update

/tmp:

/usr:
drwx--x--x   2 root     root         1024 Nov  1 15:39 bin
drwx--x--x   2 root     root         1024 Nov  1 15:39 lib
drwx--x--x   3 root     root         1024 Nov  1 15:39 man
drwx--x--x   2 root     root         1024 Nov  1 15:39 sbin
drwx--x--x   3 root     root         1024 Nov  1 15:39 share
lrwxrwxrwx   1 root     root           10 Nov  1 15:39 tmp -> ../var/tmp
```

```
/usr/bin:
-rwx--x--x   1 root     root        37164 Nov  1 15:39 afio
-rwx--x--x   1 root     root         5044 Nov  1 15:39 chroot
-rwx--x--x   1 root     root        10656 Nov  1 15:39 cut
-rwx--x--x   1 root     root        63652 Nov  1 15:39 diff
-rwx--x--x   1 root     root        12972 Nov  1 15:39 du
-rwx--x--x   1 root     root        56552 Nov  1 15:39 find
-r-x--x--x   1 root     root         6280 Nov  1 15:39 free
-rwx--x--x   1 root     root         7680 Nov  1 15:39 head
-rwx--x--x   1 root     root         8504 Nov  1 15:39 id
-r-sr-xr-x   1 root     bin          4200 Nov  1 15:39 passwd
-rwx--x--x   1 root     root        14856 Nov  1 15:39 tail
-rwx--x--x   1 root     root        19008 Nov  1 15:39 tr
-rwx--x--x   1 root     root         7160 Nov  1 15:39 wc
-rwx--x--x   1 root     root         4412 Nov  1 15:39 whoami

/usr/lib:
lrwxrwxrwx   1 root     root           17 Nov  1 15:39 libncurses.so.4 -> libncurses.so.4.2
-rw-r-r---   1 root     root       260474 Nov  1 15:39 libncurses.so.4.2

/usr/sbin:
-r-x--x--x   1 root     root        13684 Nov  1 15:39 fuser
-rwx--x--x   1 root     root         3876 Nov  1 15:39 mklost+found

/usr/share:
drwx--x--x   4 root     root         1024 Nov  1 15:39 terminfo

/usr/share/terminfo:
drwx--x--x   2 root     root         1024 Nov  1 15:39 l
drwx--x--x   2 root     root         1024 Nov  1 15:39 v

/usr/share/terminfo/l:
-rw-------   1 root     root         1552 Nov  1 15:39 linux
-rw-------   1 root     root         1516 Nov  1 15:39 linux-m
-rw-------   1 root     root         1583 Nov  1 15:39 linux-nic

/usr/share/terminfo/v:
-rw-------   2 root     root         1143 Nov  1 15:39 vt100
-rw-------   2 root     root         1143 Nov  1 15:39 vt100-am

/var:
drwx--x--x   2 root     root         1024 Nov  1 15:39 log
drwx--x--x   2 root     root         1024 Nov  1 15:39 run
drwx--x--x   2 root     root         1024 Nov  1 15:39 tmp

/var/log:
-rw-------   1 root     root            0 Nov  1 15:39 wtmp

/var/run:
-rw-------   1 root     root            0 Nov  1 15:39 utmp

/var/tmp:
```

D. 유틸리티 디스크 견본

```
total 579
-rwxr-xr-x   1 root     root        42333 Jul 28 19:05 cpio
-rwxr-xr-x   1 root     root        32844 Aug 28 19:50 debugfs
-rwxr-xr-x   1 root     root       103560 Jul 29 21:31 elvis
-rwxr-xr-x   1 root     root        29536 Jul 28 19:04 fdisk
-rw-r-r---   1 root     root       128254 Jul 28 19:03 ftape.o
-rwxr-xr-x   1 root     root        17564 Jul 25 03:21 ftmt
-rwxr-xr-x   1 root     root        64161 Jul 29 20:47 grep
-rwxr-xr-x   1 root     root        45309 Jul 29 20:48 gzip
-rwxr-xr-x   1 root     root        23560 Jul 28 19:04 insmod
-rwxr-xr-x   1 root     root          118 Jul 28 19:04 lsmod
lrwxrwxrwx   1 root     root            5 Jul 28 19:04 mt -> mt-st
-rwxr-xr-x   1 root     root         9573 Jul 28 19:03 mt-st
lrwxrwxrwx   1 root     root            6 Jul 28 19:05 rmmod -> insmod
-rwxr-xr-x   1 root     root       104085 Jul 28 19:05 tar
lrwxrwxrwx   1 root     root            5 Jul 29 21:35 vi -> elvis
```

주석

1. 여기에 제시된 디렉토리 구조는 루트디스켓에 해당되는 것만 적은 것입니다. 실제의 리눅스 시스템은 보다 복잡하고 세련된 디렉토리구조에 관한 규약을 가지고 있습니다. 이를 파일시스템 계층 표준(FHS, Filesystem Hierarchy Standard) (http://www.pathname.com/fhs/2.0/fhs-toc.html)이라 부르는데, 요컨대 각 파일들을 어느 디렉토리에 두어야 하는가에 대한 규약입니다).