Linux From Scratch

Version 9.1

2020년 3월 1일 발행

저자: Gerard Beekmans
관리 편집자: Bruce Dubbs
역자: Jongmin Kim

Linux From Scratch: Version 9.1 : 2020년 3월 1일 발행

지은이 저자: Gerard Beekmans, 관리 편집자: Bruce Dubbs, 그리고 역자: Jongmin Kim

차례

서론

머리말

리눅스를 배우고 이해하기 위한 나의 여정은 1998년에 시작되었다. 나는 내 첫 리눅스 배포판을 설치하자마자 리눅스 이면에 숨겨진 전체적인 개념과 철학에 매료되었다.

한 가지 일을 해내는 데는 항상 많은 방법이 있다. 리눅스 배포판도 마찬가지라 할 수 있다. 수년에 걸쳐 훌륭한 배포판들이 많이 존재해왔다. 어떤 것은 여전히 존재하고, 일부는 다른 무언가로 변화했지만 나머지는 기억 너머로 사라졌다. 그것들은 모두 각각의 사용자의 수요에 맞도록 다르게 동작한다. 같은 최종 목표를 달성하기 위한 다양한 방법들이 존재하기 때문에, 나는 더는 누군가의 배포판에 종속될 필요가 없다는 것을 깨닫기 시작했다. 리눅스를 만나기 전까지 우리는 그저 다른 운영 체제의 문제점들을 선택의 여지 없이 참아왔다. 당신이 좋든 싫든 늘 그래 왔다. 리눅스와 함께 선택의 개념이 생겨나기 시작했다. 무언가가 마음에 들지 않으면 자유롭게 수정할 수 있었으며, 심지어 그렇게 하도록 권장받았다.

나는 여러 배포판을 써봤지만 그 무엇으로도 결정할 수 없었다. 그것들은 그 자체로 훌륭한 시스템이었다. 더 이상 옳고 그름의 문제가 아닌 개인의 취향에 달린 것이었다. 그 많은 선택지를 보니 어느 배포판도 내게 완벽한 배포판이 될 것 같지 않았다. 그래서 나는 완벽하게 내 입맛에 맞는 나만의 리눅스 시스템을 만들기 시작했다.

나만의 시스템을 만들기 위해, 컴파일된 바이너리 패키지를 사용하는 대신 모든 것을 소스 코드부터 컴파일하기로 결정했다. 이 “완벽한” 리눅스 시스템은 여러 배포판들의 단점을 제외하고 장점만 취할 것이었다. 처음엔, 이 계획이 다소 어려워 보였다. 나는 그러한 시스템을 구축할 수 있다는 생각에 계속 전념했다.

종속성 순환이나 컴파일 에러같은 문제들을 겪고나서, 마침내 커스텀 리눅스 시스템을 구축했다. 당시의 다른 리눅스 배포판과 마찬가지로 완벽하게 작동하고 완벽하게 사용할 수 있었다. 그러나 이것은 내가 직접 만든 것이었다. 그런 시스템을 스스로 구성해보니 매우 만족스러웠다. 이보다 더 나은 것은 각각의 소프트웨어마저 직접 만드는 것이었을 것이다. 이것은 차선책이었다.

리눅스 커뮤니티의 다른 구성원들에게 나의 목표와 경험을 공유하면서, 이러한 아이디어에 지속적인 관심이 있다는 것이 분명해졌다. 이 맞춤형 리눅스 시스템은 사용자별 요구 사항을 충족시킬 뿐만 아니라 프로그래머와 시스템 관리자가 (기존) 리눅스 기술을 향상시킬 수 있는 이상적인 학습 기회가 된다는 것이 분명해졌다. 이렇게 넓어진 관심으로부터 Linux From Scratch 프로젝트가 탄생했다.

이 Linux From Scratch 책은 그 프로젝트의 핵심이다. 이것은 당신이 자신만의 시스템을 설계하고 구축하는 데 필요한 배경과 지침을 제공한다. 이 책은 올바르게 작동하는 시스템을 만들어 낼 수 있는 템플릿을 제공하지만, 당신은 자유자재로 지시를 변경할 수 있다. 이것이 바로 이 프로젝트의 중요한 부분이다. 당신은 통제권을 가지고 있다; 우리는 단지 당신만의 여행을 시작할 수 있도록 도움의 손길을 뻗어줄 뿐이다.

여러분만의 Linux From Scratch 시스템에서 즐거운 시간을 보내고, 진정한 자신만의 시스템을 가짐으로써 얻을 수 있는 수많은 혜택을 누리길 진심으로 바란다.

--
Gerard Beekmans
gerard@linuxfromscratch.org

독자에게

당신이 이 책을 읽고 싶어할 만한 많은 이유가 있다. 많은 사람들이 제기하는 질문 중 하나는 “기존에 있던 리눅스 시스템을 다운로드해서 설치하면 되는데 왜 일일이 수동으로 구축해야 하는 번거로움을 감수하는가?”이다.

이 프로젝트가 존재하는 한 가지 중요한 이유는 리눅스 시스템이 어떻게 작동하는지 내부에서 배울 수 있도록 돕기 위함이다. LFS 시스템을 구축하면 무엇이 리눅스를 움직이게 하는지, 그리고 어떻게 함께 작동하고 서로 의존하는지를 깨닫는데 도움이 된다. 이 경험에서 얻을 수 있는 가장 좋은 것 중 하나는 리눅스 시스템을 자신의 고유한 목적에 맞게 사용자 정의할 수 있는 능력이다.

LFS의 또 다른 주요 장점은 다른 누군가가 구성한 리눅스 배포판에 의존하지 않고 시스템 전반을 더 많이 제어할 수 있다는 점이다. LFS에서는 직접 운전석에 앉아 시스템의 모든 면을 통제할 수 있다.

LFS는 매우 가벼운 리눅스 시스템을 만들 수 있다. 정규 배포판을 설치할 때는, 절대 사용하지도 않거나 이해할 수 없는 많은 프로그램들을 설치하도록 강요 받는다. 이 프로그램들은 자원을 낭비한다. 오늘날의 하드 드라이브와 CPU에서는 이제 그런 걱정할 필요 없다고 주장할 지도 모르지만 종종 선택의 여지가 없다면 여전히 용량의 제약을 받는다. 부팅 가능한 CD, USB 스틱이나 임베디드 시스템에 대해 생각해 보라. 그 곳들이 LFS가 유용한 분야다.

맞춤형 리눅스 시스템의 또 다른 장점은 보안성이다. 소스 코드에서 전체 시스템을 컴파일하여 모든 것을 감시하고 원하는 모든 보안 패치를 적용할 수 있다. 보안 허점을 고치는 바이너리 패키지를 다른 사람이 컴파일하는 것을 더는 기다릴 필요가 없다. 패치를 검사하여 직접 구현하지 않는 한 새로운 바이너리 패키지가 올바르게 빌드되어 문제를 적절히 해결했다는 보장도 없다.

Linux From Scratch의 목표는 완전하고 사용 가능한 수준의 기반 시스템을 구축하는 것이다. 만약 당신이 처음부터 당신만의 리눅스 시스템을 구축하기를 원하지 않더라도, 이 책의 정보들은 도움이 될 수 있다.

여러분만의 LFS 시스템을 구축해야 할 다른 좋은 이유가 너무 많아서 그것들을 여기에 일일이 나열할 수 없다. 결국, 학습이 그들 중 단연코 가장 강력한 이유일 것이다. LFS 공부를 계속하면서 정보와 지식이 진정으로 가져다주는 힘을 발견하게 될 것이다.

LFS 대상 아키텍쳐

LFS의 주요 대상 아키텍처는 AMD/Intel x86(32비트)과 x86_64(64비트) CPU다. 한편으론 이 책에 수록된 절차들에서 어느 정도 수정을 거치면 파워 PC와 ARM CPU에서도 작동하는 것으로 알려져 있다. 이러한 CPU 중 하나를 활용하는 시스템을 구축하기 위해서는 다음 페이지에 있는 것 외에, 우분투, 레드햇/페도라, SusE, 또는 당신이 가지고 있는 아키텍처를 대상으로 하는 다른 배포판같은 기존 리눅스 시스템이 필요하다. 32비트 버전도 64비트 AMD/Intel 컴퓨터에 설치하여 호스트 시스템으로 사용할 수 있다는 점을 참고하라.

LFS 구축의 경우, 32비트 시스템에 비해 64비트 시스템이 얻는 장점은 크지 않다. 예를 들면, 4코어를 사용하는 인텔 코어 i7-4790 CPU 기반 시스템에 대한 LFS-9.1 빌드 테스트 통계는 다음과 같이 측정되었다.

아키텍쳐	빌드 시간	빌드 사이즈
32-bit	239.9 분	3.6 GB
64-bit	233.2 분	4.4 GB

보다시피 동일한 하드웨어에서 64비트 빌드는 32비트 빌드보다 고작 3% 빠르지만 22% 더 크다. LFS를 LAMP 서버 또는 방화벽으로 사용하려는 경우 32비트 CPU로 충분할 수 있다. 반면, 현재 BLFS의 여러 패키지는 빌드/실행하는데에 4GB의 RAM을 필요로 하므로 LFS를 데스크톱으로 사용할 계획이라면 64비트 시스템으로 구축할 것을 LFS 저자들은 권장한다.

LFS의 결과물로 나온 기본 64비트 빌드는 “순수한” 64비트 시스템으로 간주된다. 즉, 64비트 실행 파일만 지원한다. “multi-lib” 시스템을 구축하려면 많은 애플리케이션을 32비트 시스템에 한 번, 64비트 시스템에 한 번 컴파일해야 한다. 이것은 기초 리눅스 시스템으로 가는데 필요한 과정을 교육 하는 목적에 방해가 될 수 있기 때문에 LFS에서는 직접적으로 다루지 않는다. 일부 LFS/BLFS 편집자는 http://www.linuxfromscratch.org/~thomas/multilib/index.html에서 multilib용 LFS 포크를 관리 중이다. 그러나 아직 갈 길이 먼 주제다.

전제 조건

LFS 시스템을 구축하는 것은 간단한 일이 아니다. 문제를 해결하고 나열된 명령을 올바르게 실행하기 위해서는 유닉스 시스템 관리에 대한 일정 수준의 기본 지식이 필요하다. 특히, 아무리 못해도 커맨드 라인(셸)을 사용하여 파일과 디렉토리를 복사하거나 이동시키고, 디렉토리 및 파일 내용을 나열하고, 현재 디렉토리를 변경할 수 있는 능력을 이미 갖추고 있어야 한다. 또한 리눅스 소프트웨어 사용 및 설치에 대한 합리적인 지식이 있을 것으로 간주한다.

LFS 책은 이러한 기본적인 수준의 기술을 최소한의 능력으로 간주하기 때문에, 여러 LFS 지원 포럼들은 이러한 부분에서 당신에게 많은 도움을 줄 수 없을 것이다. 당신의 그러한 기초지식에 관한 질문들에는 답변이 오지 않거나 그저 LFS 필독 사항 리스트만 추천 받게 될 수 있다.

LFS 시스템을 빌드하기 전에, 다음을 읽어보기 바란다:

- Software-Building-HOWTO http://www.tldp.org/HOWTO/Software-Building-HOWTO.html

  리눅스에서 “일반적인” 유닉스 소프트웨어 패키지를 빌드하고 설치하기 위한 포괄적인 안내서다. 비록 작성된 지 오래 되었지만, 여전히 소프트웨어를 만들고 설치하는 데 필요한 기본적인 기술들을 잘 요약해 준다.
- Beginner's Guide to Installing from Source http://moi.vonos.net/linux/beginners-installing-from-source/

  이 안내서는 소스 코드에서부터 소프트웨어를 빌드하기 위한 기본 스킬과 테크닉에 대해 잘 요약해준다.

LFS와 표준

LFS의 구조는 가능한 한 리눅스 표준을 따른다. 주요 표준들은 다음과 같다:

- POSIX.1-2008.
- Filesystem Hierarchy Standard (FHS) Version 3.0
- Linux Standard Base (LSB) Version 5.0 (2015)

  LSB에는 코어, 데스크탑, 런타임 언어 및 이미징의 4가지 별도의 표준이 있다. 일반적인 조건 외에 아키텍쳐별 요구 사항도 있다. 또한 시험적으로 적용할 수 있는 영역은 Gtk3와 그래픽스 두 가지가 있다. LFS는 가능한 한 이전 절에서 언급한 아키텍쳐에 부합할 것이다.

참고

LSB의 요구사항에 동의하지 않는 사람들도 많다. 이를 정의하는 주된 목적은 독점 소프트웨어가 호환 시스템에서 제대로 설치되고 실행될 수 있도록 하는 것이다. LFS는 소스 기반이기 때문에 사용자는 원하는 패키지를 완전히 제어할 수 있으며 많은 사용자가 LSB에 의해 지정된 일부 패키지를 설치하지 않기도 한다.

LSB 인증 시험에 합격할 수 있는 완전한 LFS 시스템을 만드는 것은 가능하지만 LFS의 범위를 벗어나는 많은 추가 패키지가 필요하다. 이 추가 패키지는 BLFS에 설치 지침이 있다.

LFS에서 제공하는 LSB 요구 사항을 만족하는 데 필요한 패키지

LSB 코어:	Bash, Bc, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib
LSB 데스크탑:	없음
LSB 런타임 언어:	Perl

LSB 이미징:	없음
LSB Gtk3와 LSB 그래픽스 (시범 적용):	없음

BLFS에서 제공하는 LSB 요구 사항을 만족하는 데 필요한 패키지

LSB 코어:	At, Batch (At의 일부), Cpio, Ed, Fcrontab, LSB-Tools, NSPR, NSS, PAM, Pax, Sendmail (또는 Postfix나 Exim), time
LSB 데스크탑:	Alsa, ATK, Cairo, Desktop-file-utils, Freetype, Fontconfig, Gdk-pixbuf, Glib2, GTK+2, Icon-naming-utils, Libjpeg-turbo, Libpng, Libtiff, Libxml2, MesaLib, Pango, Xdg-utils, Xorg
LSB 런타임 언어:	Python, Libxml2, Libxslt
LSB 이미징:	CUPS, Cups-filters, Ghostscript, SANE
LSB Gtk3와 LSB 그래픽스 (시범 적용):	GTK+3

LFS나 BLFS에서 제공하지 않는, LSB 요구 사항을 만족하는 데 필요한 패키지

LSB 코어:	없음
LSB 데스크탑:	Qt4 (단 Qt5는 제공됨)
LSB 런타임 언어:	없음
LSB 이미징:	없음
LSB Gtk3와 LSB 그래픽스 (시범 적용):	없음

책의 패키지에 대한 설명

앞서 말한 바와 같이, 완전하고 사용 가능한 기초 수준의 시스템을 구축하는 것이 LFS의 목표다. 여기에는 사용자의 선택에 따라 보다 완전한 시스템을 사용자 정의하기 위한 비교적 최소한의 기반을 제공하면서 자체 복제에 필요한 모든 패키지가 포함된다. 이것은 LFS가 가능한 가장 작은 시스템이라는 것을 의미하지 않는다. 엄격하게 요구되지 않는 몇 가지 중요한 패키지가 포함되어 있다. 아래 목록은 책에 있는 각 패키지에 대한 설명이다.

- Acl

  접근 제어 목록(Access Control Lists)을 관리하는 도구가 포함되어 있으며, 파일과 디렉토리에 대한 보다 세분화된 재량적 접근 권한을 정의하는 데 사용된다.

- Attr

  파일 시스템 객체의 확장된 속성을 관리하기 위한 프로그램

- Autoconf

  개발자 템플릿에서 소스 코드를 자동으로 구성할 수 있는 셸 스크립트 생성 프로그램이 포함되어 있다. 빌드 절차를 업데이트한 후 패키지를 다시 빌드해야 하는 경우가 많다.

- Automake

  템플릿으로부터 Make 파일을 생성하는 프로그램을 포함하고 있다. 빌드 절차를 업데이트한 후 패키지를 다시 빌드해야 하는 경우가 많다.

- Bash

이 패키지는 시스템에 Bourne Shell 인터페이스를 제공하는 LSB 코어 요구 사항이다. 다른 셸 패키지들 사이에서 채택된 것은 보편적으로 쓰이고 기본 셸 기능을 넘어서는 광범위한 기능 때문이다.

- Bc

  임의의 정밀 숫자 처리 언어를 제공한다. 리눅스 커널을 구축할 때 필요한 패키지다.

- Binutils

  오브젝트 파일들을 처리하기 위한 링커, 어셈블러 및 기타 도구가 포함되어 있다. 이 패키지의 프로그램은 LFS 시스템의 대부분의 패키지, 그리고 이후에도 컴파일하기 위해 필요하다.

- Bison

  여러 다른 LFS 프로그램을 빌드하는 데 필요한 Yacc(Yet Another Compiler Compiler Compiler)의 GNU 버전이 포함되어 있다.

- Bzip2

  파일을 압축하고 해제하는 프로그램이 포함되어 있다. 많은 LFS 패키지의 압축을 푸는 데 필요하다.

- Check

  다른 프로그램을 위한 테스트 도구가 포함되어 있다.

- Coreutils

  파일 및 디렉터리를 보고 조작하는 데 필수적인 프로그램이 다수 포함되어 있다. 이 프로그램들은 명령 줄 파일 관리에 쓰이며 LFS의 모든 패키지의 설치 절차에 필요하다.

- DejaGNU

  다른 프로그램을 테스트하기 위한 프레임워크를 포함하고 있다. 임시 툴체인에만 설치된다.

- Diffutils

  파일 또는 디렉터리 간의 차이를 보여주는 프로그램이 포함되어 있다. 이 프로그램은 패치를 만드는 데 사용할 수 있으며, 많은 패키지의 빌드 절차에도 사용된다.

- E2fsprogs

  ext2, ext3, ext4 파일 시스템을 처리하는 도구가 포함되어 있다. 이들은 리눅스가 지원하는 가장 보편적이고 철저하게 테스트된 파일 시스템이다.

- Eudev

  이 패키지는 장치 관리자이다. 디바이스가 시스템에서 추가 또는 제거될 때 /dev 디렉토리의 항목을 동적으로 제어한다.

- Expat

  비교적 작은 XML 구문 분석 라이브러리가 포함되어 있다. XML::Parser Perl 모듈에 필요하다.

- Expect

  다른 대화형 프로그램과 스크립트로 통신하기 위한 프로그램이 포함되어 있다. 일반적으로 다른 패키지들을 테스트하는데 사용된다. 임시 툴체인에만 설치된다.

- File

  지정된 파일들의 유형을 구분하는 유틸리티가 포함되어 있다. 몇몇 꾸러미를 빌드하는 데 필요하다.

- Findutils

파일 시스템에서 파일을 찾는 프로그램이 포함되어 있다. 많은 패키지의 빌드 스크립트에 사용된다.

- Flex

  텍스트의 패턴을 인식하는 프로그램을 생성하는 도구가 포함되어 있다. 렉스 프로그램(lexical Analyzer)의 GNU 버전이다. 여러 LFS 패키지들을 빌드하는 데 필요하다.

- Gawk

  텍스트 파일을 조작하는 프로그램을 포함하고 있다. GNU 버전의 awk(Aho-Weinberg-Kernighan)이다. 다른 많은 패키지의 빌드 스크립트에 사용된다.

- Gcc

  GNU 컴파일러 모음이다. C와 C++ 컴파일러뿐만 아니라 LFS에 의해 빌드되지 않는 여러 다른 컴파일러도 포함하고 있다.

- GDBM

  GNU 데이터베이스 관리자 라이브러리를 포함한다. 다른 LFS 패키지인 Man-DB에서 사용된다.

- Gettext

  수많은 패키지의 다국어 지원을 위한 도구와 라이브러리가 포함되어 있다.

- Glibc

  메인 C 라이브러리를 포함하고 있다. 리눅스 프로그램은 이것 없이는 실행되지 않을 것이다.

- GMP

  임의 정밀 산술에 유용한 함수를 제공하는 수학 라이브러리가 포함되어 있다. GCC를 빌드할 때 필요하다.

- Gperf

  키 세트로부터 완벽한 해시함수를 생성하는 프로그램이 포함되어 있다. Eudev를 위해 필요하다.

- Grep

  파일들을 검색하는 프로그램이 포함되어 있다. 이 프로그램은 대부분의 패키지의 빌드 스크립트에서 사용된다.

- Groff

  텍스트를 처리하고 서식화하는 프로그램이 포함되어 있다. 이 프로그램들의 중요한 기능 중 하나는 Man 페이지 서식을 만드는 것이다.

- GRUB

  이 패키지는 the Grand Unified Boot Loader이다. 여러 부트 로더 중 하나지만 가장 유연하다.

- Gzip

  파일을 압축하고 해제하는 프로그램이 포함되어 있다. LFS와 다른 패키지의 압축을 푸는 데 필요하다.

- Iana-etc

  네트워크 서비스와 프로토콜에 대한 데이터를 제공한다. 적절한 네트워킹 기능을 활성화하는 데 필요하다.

- Inetutils

  기본 네트워크 관리 프로그램이 포함되어 있다.

- Intltool

  소스 파일에서 변환 가능한 문자열을 추출하는 도구가 포함되어 있다.

- IProute2

  기본 및 고급 IPv4 및 IPv6 네트워킹을 위한 프로그램이 포함되어 있다. IPv6 기능을 위해 다른 일반 네트워크 도구 패키지(net-tools) 사이에서 채택되었다.

- Kbd

  키-테이블 파일, 미국 이외의 키보드를 위한 키보드 유틸리티 및 다수의 콘솔 글꼴이 포함되어 있다.

- Kmod

  리눅스 커널 모듈을 관리하는 데 필요한 프로그램이 포함되어 있다.

- Less

  파일을 볼 때 위아래로 스크롤할 수 있는 매우 멋진 텍스트 파일 뷰어가 포함되어 있다. Man페이지를 볼 때도 Man-DB에서 사용된다.

- Libcap

  리눅스 커널에서 사용할 수 있는 POSIX 1003.1e 기능에 대한 사용자 공간 인터페이스를 구현한다.

- Libelf

  elfutils 프로젝트는 ELF 파일과 DWARF 데이터를 위한 라이브러리와 도구를 제공한다. 이 패키지의 대부분의 유틸리티는 다른 패키지에서도 제공되지만, 이 패키지의 라이브러리는 기본(가장 효율적인) 구성을 사용하여 리눅스 커널을 구축하기 위해 필요하다.

- Libffi

  다양한 호출 규약에 대해 포팅 가능한 고수준(high level) 프로그래밍 인터페이스를 구현한다. 어떤 프로그램은 어떤 인자를 함수에 전달해야 하는지를 컴파일할 때 알지 못할 수 있다. 예를 들어, 인터프리터에는 함수를 호출하는 데 사용되는 인수의 수와 유형들이 런타임에 주어진다. Libffi는 그러한 인터프리터 프로그램에서 컴파일된 코드로의 다리 역할을 한다.

- Libpipeline

  유연하고 편리한 방법으로 하위 프로세스의 파이프라인을 조작하기 위한 라이브러리를 포함하고 있다. Man-DB 패키지에 필요하다.

- Libtool

  GNU 일반 라이브러리 지원 스크립트가 포함되어 있다. 일관적으로 공유 라이브러리를 사용하는 복잡성을 포팅 가능한 인터페이스로 래핑한다. 다른 LFS 패키지의 테스트 스위트에 필요하다.

- Linux Kernel

  운영 체제이다. GNU/리눅스 환경의 리눅스이다.

- M4

  다른 프로그램의 빌드 도구로 유용한 범용 텍스트 매크로 프로세서가 포함되어 있다.

- Make

  패키지 빌드를 위한 프로그램을 포함되어 있다. 거의 모든 LFS 패키지에 필요하다.

- Man-DB

  Man 페이지를 검색하고 볼 수 있는 프로그램이 포함되어 있다. 우수한 다국어 지원 때문에 man 패키지 대신 채택되었다. Man 프로그램을 지원한다.

- Man-pages

이 패키지에는 기본적인 리눅스 맨 페이지의 실제 내용이 들어 있다.

- Meson

  소프트웨어 빌드를 자동화하는 소프트웨어 도구를 제공한다. Meson의 주요 목표는 소프트웨어 개발자들이 그들의 빌드 시스템을 구성하는 데 필요한 시간을 최소화하는 것이다.

- MPC

  복잡한 숫자의 산수를 위한 함수가 포함되어 있다. gcc에 필요하다.

- MPFR

  다중 정밀 산술 함수가 포함되어 있다. gcc에 필요하다.

- Ninja

  속도에 초점을 둔 소형 빌드 시스템을 포함하고 있다. 더 높은 수준의 빌드 시스템에 의해 생성된 입력 파일을 받으며, 가능한 빨리 빌드를 수행하도록 설계되었다.

- Ncurses

  문자 화면의 터미널 독립적 처리를 위한 라이브러리가 포함되어 있다. 종종 메뉴 시스템을 위한 커서 제어를 위해 사용된다. LFS의 많은 패키지에 필요하다.

- Openssl

  암호화와 관련된 관리 도구와 라이브러리를 제공한다. 리눅스 커널을 포함한 다른 패키지의 암호화 기능을 제공하는 데 유용하다.

- Patch

  일반적으로 diff 프로그램에 의해 생성된 패치 파일을 적용하여 파일을 수정하거나 만드는 프로그램이 포함되어 있다. 몇 개의 LFS 패키지에 대한 빌드 절차에 필요하다.

- Perl

  런타임 언어 PERL의 인터프리터이다. 여러 LFS 패키지의 설치 및 테스트 스위트를 위해 필요하다.

- Pkg-config

  설치된 라이브러리 또는 패키지에 대한 메타 데이터를 반환하는 프로그램을 제공한다.

- Procps-NG

  프로세스 모니터링 프로그램이 포함되어 있다. 이러한 프로그램은 시스템 관리에 유용하며 LFS 부트스크립트에 사용되기도 한다.

- Psmisc

  실행 중인 프로세스들의 정보를 표시하는 프로그램이 포함되어 있다. 시스템 관리에 유용하다.

- Python 3

  코드 가독성을 강조하는 디자인 철학이 담긴 인터프리터 언어를 제공한다.

- Readline

  명령줄 편집 및 기록 기능을 제공하는 라이브러리 모음이다. Bash에 쓰인다.

- Sed

  텍스트 편집기에서 열지 않고도 텍스트를 편집할 수 있다. 대부분의 LFS 패키지의 구성 스크립트에도 필요하다.

- Shadow

  안전한 방법으로 비밀번호를 관리하는 프로그램이 포함되어 있다.

- Sysklogd

  비정상적인 이벤트가 발생할 때 커널 또는 데몬 프로세스에 의해 출력된 시스템 메시지를 로깅하는 프로그램이 포함되어 있다.

- Sysvinit

  리눅스 시스템의 다른 모든 프로세스의 부모 프로세스인 init 프로그램을 제공한다.

- Tar

  LFS에 사용되는 거의 모든 패키지의 보관 및 추출 기능을 제공한다.

- Tcl

  LFS 패키지의 많은 테스트 스위트에서 사용되는 Tool Command Language가 포함되어 있다. 임시 툴체인에만 설치된다.

- Texinfo

  info 페이지를 읽고, 쓰고, 변환하는 프로그램이 포함되어 있다. 많은 LFS 패키지의 설치 절차에 사용된다.

- Util-linux

  다양한 유틸리티 프로그램이 포함되어 있다. 그 중에는 파일 시스템, 콘솔, 파티션 및 메시지를 처리하는 유틸리티가 있다.

- Vim

  이 패키지에는 편집기가 들어 있다. 고전 vi 에디터와의 호환성과 엄청나게 다양하고 강력한 능력 때문에 채택되었다. 편집기는 많은 사용자에게 매우 개인적인 취향일 뿐, 원한다면 다른 편집기로 대체할 수 있다.

- XML::Parser

  Expat과 접속하는 Perl 모듈이다.

- XZ Utils

  파일을 압축하고 해제하는 프로그램이 포함되어 있다. 일반적으로 사용할 수 있는 가장 높은 압축률을 제공하며, XZ나 LZMA 형식의 패키지 압축을 푸는 데 유용하다.

- Zlib

  일부 프로그램에서 사용하는 압축 및 압축 해제 루틴이 포함되어 있다.

- Zstd

  일부 프로그램에서 사용하는 압축 및 압축 해제 루틴이 포함되어 있다. 높은 압축율과 매우 광범위한 압축/속도 절충 기능을 제공한다.

타이포그래피

따르기 쉽도록 하기 위해, 이 책 전반에 걸쳐 사용되는 몇 가지 활자 규칙이 있다. 이 절에는 Linux From Scratch에서 찾을 수 있는 활자 형식의 몇 가지 예가 수록되어 있다.

```
./configure --prefix=/usr
```

이 형식의 텍스트는 옆에 달리 명시되지 않는 한 보이는 대로 정확하게 입력하라. 또한 설명하는 부분에서는 어떤 명령어를 설명하고 있는지 가리키기 위해 사용된다.

어떤 경우에는 한 논리적 명령이 두 줄 이상으로 길어져 행 끝에 백슬래시가 있는 경우도 있다.

```
CC="gcc -B/usr/bin/" ../binutils-2.18/configure ₩
  --prefix=/tools --disable-nls --disable-werror
```

백슬래시 뒤에 바로 줄바꿈 후 이어적어야 한다는 점을 유의하라. 스페이스나 탭 문자와 같은 공백 문자는 잘못된 결과를 낳을 것이다.

```
install-info: unknown option '--dir-file=/mnt/lfs/usr/info/dir'
```

이 형식의 텍스트(고정폭 텍스트)는 화면 출력, 일반적으로 실행된 명령의 결과를 보여준다. 이 형식은 /etc/ld.so.conf와 같이 파일 이름을 표시하는 데도 사용된다.

Emphasis

이 형식의 텍스트는 이 책에서 여러 가지 용도로 사용된다. 중요한 부분이나 항목을 강조하는 것이 주요 목적이다.

http://www.linuxfromscratch.org/

이 형식은 LFS 커뮤니티 내의 하이퍼링크와 외부 페이지에 모두 사용된다. HowTO, 다운로드 위치 및 웹 사이트를 포함한다.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
......
EOF
```

이 형식은 구성 파일을 만들 때 사용된다. 첫 번째 명령은 다음 줄에 무엇이 입력되든 EOF(End Of File) 시퀀스를 마주 할 때까지 입력된 내용으로 $LFS/etc/group 파일을 생성하도록 시스템에 지시한다. 따라서 일반적으로 이 부분 전체를 보이는 대로 입력하면 된다.

<REPLACED TEXT>

이 형식은 그대로 입력하지 않는 텍스트나 복사/붙여넣기 작업을 위한 텍스트를 묶는 데 사용된다.

[OPTIONAL TEXT]

이 형식은 선택 사항인 텍스트를 묶는 데 사용된다.

passwd(5)

이 형식은 특정 매뉴얼(man) 페이지를 가리키는 데 사용된다. 괄호 안의 숫자는 매뉴얼 내부의 특정 섹션을 나타낸다. 예를 들어, passwd는 두 개의 Man 페이지를 가지고 있다. LFS 설치 지침에 따라 이 두 개의 Man 페이지는 /usr/share/man/man1/passwd.1 과 /usr/share/man/man5/passwd.5에 위치한다. 이 책에서 passwd(5)라 하면, 그것은 /usr/share/man/man5/passwd.5를 가리킨다. man passwd는 “passwd”와 일치하는 첫 번째 Man 페이지, /usr/share/man/man1/passwd.1를 출력한다. 이 예시에서는, 가리켜진 특정 페이지를 읽으려면 man 5 passwd를 실행해야 한다. 대부분의 man 페이지에는 서로 다른 섹션에 중복된 페이지 이름이 없다는 점을 참고하라. 따라서 일반적으론 man <program name>으로 충분하다.

책의 짜임새

이 책은 다음과 같은 부분으로 나뉜다.

1부 - 소개

1부에선 LFS 설치를 진행하는 방법에 대한 몇 가지 중요한 참고 사항을 설명한다. 이 절은 또한 책에 대한 메타 정보도 제공한다.

2부 - 빌드를 위한 준비

2부는 파티션 생성, 패키지 다운로드, 임시 도구 컴파일 등 빌드 절차를 준비하는 방법을 설명한다.

3부 - LFS 시스템 빌드

3부에선 LFS 시스템 빌드를 설명한다—모든 패키지를 하나씩 컴파일 및 설치, 부트 스크립트 설정, 커널 설치. 그 결과물인 리눅스 시스템은 다른 소프트웨어를 빌드하여 원하는 대로 시스템을 확장할 수 있는 기틀이다. 이 책의 마지막 부분에는 설치된 모든 프로그램, 라이브러리, 중요한 파일들을 나열한 쉬운 참조 자료들이 있다.

정오표

LFS 시스템을 만드는 데 사용되는 소프트웨어는 지속적으로 업데이트되며 향상되고 있다. 보안 경고와 버그 수정은 LFS 책이 공개된 후에도 나올 수 있다. 보안 취약점이나 기타 버그를 수정하기 위해 이 LFS 내 패키지의 버전 또는 절차를 변경해야 하는지 확인하려면 빌드를 진행하기 전에 http://www.linuxfromscratch.org/lfs/errata/9.1/을 방문하라. 명시된 변경 사항들을 참고하여 책의 관련 부분에 적용해서 LFS 시스템 구축을 진행하라

부 I. 소개

1장. 들어가기 전에

1.1. LFS 시스템을 빌드하는 방법

LFS 시스템은 이미 설치된 Linux 배포판(데비안, OpenMandriva, 페도라 또는 OpenSUSE 등)을 사용하여 구축할 것이다. 이 기존의 리눅스 시스템(호스트)은 컴파일러, 링커, 셸 등 새로운 시스템을 구축하는 데 필요한 프로그램을 제공하는 출발점으로 활용된다. 이러한 도구를 사용하려면 배포판을 설치할 때 “개발(development)” 옵션을 선택하라.

컴퓨터에 따로 배포판을 설치하는 대신 상용 배포판의 라이브CD를 사용할 수도 있다.

이 책의 2장에서는 새로운 리눅스 네이티브 파티션과 파일 시스템을 만드는 방법을 설명한다. 그 새 파티션이 새로운 LFS 시스템을 컴파일하고 설치할 곳이다. 3장에서는 LFS 시스템을 구축하기 위해 어떤 패키지와 패치를 다운로드해야 하는지, 그리고 이를 새 파일 시스템에 저장하는 방법을 설명한다. 4장에서는 적절한 작업 환경의 설정을 논한다. 5장과 그 이후 작업을 시작하기 전에 알아야 할 몇 가지 중요한 문제를 설명하므로 4장을 주의 깊게 읽기 바란다.

5장은 6장에서 실제 시스템을 구축하는 데 사용되는 기본 개발 도구(또는 툴체인)를 만들 다수의 패키지의 설치를 설명한다. 이러한 패키지 중 일부는 순환 의존성(예를 들어, 컴파일러를 컴파일하기 위해 컴파일러가 필요한 상황)을 해결하기 위해 필요하다.

5장에서는 Binutils와 GCC를 포함한 툴체인의 1단계를 빌드하는 방법도 설명한다 (다시말해 이 두 가지 핵심 패키지는 추후 재설치할 것이다). 다음 단계는 C 라이브러리인 Glibc를 빌드하는 것이다. Glibc는 앞서 구축된 툴체인 프로그램으로 빌드된다. 그리고 나서, 툴체인의 두 번째 단계를 빌드한다. 여기서, 툴체인은 새로 빌드된 Glibc과 함께 동적으로 링크될 것이다. 나머지 5장 패키지는 이 2단계 툴체인을 사용하여 제작된다. 이 작업이 완료되면 실행 중인 커널을 제외하고 LFS 설치 프로세스는 더 이상 호스트 시스템에 종속되지 않는다.

호스트 시스템에서 새 시스템을 분리하려는 이러한 노력은 불필요해 보일지 모른다. 이 작업을 수행하는 이유에 대한 전체적인 기술적 설명은 5.2절. “툴체인 기술 노트”에 수록되어 있다.

6장에서는 완전한 LFS 시스템을 구축한다. chroot(root 변경) 프로그램은 가상 환경에 들어가 루트 디렉토리가 LFS 파티션으로 설정될 새 셸을 시작하는 데 사용된다. 이는 재부팅 후 커널로 하여금 LFS 파티션을 루트 파티션으로 마운트하도록 지시하는 것과 매우 흡사하다. 이 시스템은 실제로 재부팅하지 않고, 대신 chroot을 사용한다. 왜냐하면 부팅 가능한 시스템을 만들려면 아직 당장은 필요하지 않은 추가 작업이 필요하기 때문이다. “chrooting”의 주요 장점은 LFS가 구축되는 동안 호스트 시스템을 계속 사용할 수 있다는 점이다. 패키지 컴파일이 완료되기를 기다리는 동안 컴퓨터를 평소처럼 계속 사용할 수 있다.

설치를 마치기 위해 7장에서 기본 시스템 구성을 설정하고, 8장에서 커널과 부트 로더를 설정한다. 9장은 이 책을 넘어 LFS 경험을 지속하는 것에 대한 정보를 담고 있다. 이 책의 절차들을 실행하고 나면 컴퓨터는 새로운 LFS 시스템으로 다시 부팅할 준비가 될 것이다.

여기까지가 전체적인 과정이다. 각 단계에 대한 자세한 정보는 다음 장과 패키지 설명에서 논의한다. LFS 모험에 나서면서 복잡해 보일 수 있는 설명들이 점차 명확해질 것이고, 모든 것이 제자리를 찾을 것이다.

1.2. 지난 릴리스 이후 새로운 기능

아래는 이 책의 이전 출간 이후에 만들어진 패키지 업데이트 목록이다.

업그레이드 됨:

•

• Bc 2.5.3

- Binutils-2.34
- Bison-3.5.2
- Check-0.14.0
- E2fsprogs-1.45.5
- Eudev-3.2.9
- Expat-2.2.9
- File-5.38
- Findutils-4.7.0
- Glibc-2.31
- GMP-6.2.0
- Grep-3.4
- IPRoute2-5.5.0
- Libcap-2.31
- Libelf-0.178 (from elfutils)
- Libffi-3.3
- Libpipeline-1.5.2
- Linux-5.5.3
- Make-4.3
- Man-DB-2.9.0
- Man-pages-5.05
- Meson-0.53.1
- Ncurses-6.2
- Ninja-1.10.0
- Openssl-1.1.1d
- Perl-5.30.1
- Python-3.8.1
- Sed-4.8
- Shadow-4.8.1
- SysVinit-2.96
- Tcl-8.6.10
- Texinfo-6.7
- Tzdata-2019c
- Util-Linux-2.35.1
- Vim-8.2.0190

추가됨:

- Zstd-1.4.4

삭제됨:

-

1.3. 변경 사항

이것은 2020년 3월 1일자 Linux From Scratch 책의 9.1 버전이다. 만약 이 책이 6개월 이상 된 것이라면, 아마도 이미 더 새롭고 더 나은 버전을 구할 수 있을 것이다. 자세한 내용은 http://www.linuxfromscratch.org/mirrors.html을 통해 미러 중 하나를 확인하라.

아래는 이전 출간 이후에 변경된 사항의 목록이다.

변경기록 사항:

- 2020-03-01
  - [bdubbs] - LFS-9.1 출간됨.
- 2020-02-14
  - [bdubbs] - bison-3.5.2로 업데이트. #4597수정됨.
- 2020-02-13
  - [bdubbs] - ncurses-6.2로 업데이트. #4596 수정됨.
  - [bdubbs] - man-pages-5.05로 업데이트. #4595 수정됨.
  - [bdubbs] - linux-5.5.3.tar.xz로 업데이트. #4592 수정됨.
- 2020-01-27
  - [bdubbs] - vim-8.2.0190로 업데이트. #4500.
  - [bdubbs] - binutils-2.34로 업데이트. #4590 수정됨.
  - [bdubbs] - glibc-2.31로 업데이트. #4589 수정됨.
  - [bdubbs] - linux-5.5.1로 업데이트. #4588 수정됨.
  - [bdubbs] - bc-2.5.3로 업데이트. #4587 수정됨.
  - [bdubbs] - iproute2-5.5.0로 업데이트. #4586 수정됨.
  - [bdubbs] - util-linux 2.35.1로 업데이트. #4560 수정됨.
- 2020-01-27
  - [bdubbs] - ninja-1.10.0.로 업데이트. #4585 수정됨.
  - [bdubbs] - check-0.14.0.로 업데이트. #4583 수정됨.
  - [bdubbs] - shadow-4.8.1.로 업데이트. #4582 수정됨.
  - [bdubbs] - meson-0.53.1.로 업데이트. #4581 수정됨.
  - [bdubbs] - linux-5.5.로 업데이트. #4580 수정됨.
  - [bdubbs] - bison-3.5.1.로 업데이트. #4579 수정됨.
- 2020-01-19
  - [bdubbs] - make-4.3.로 업데이트. #4578 수정됨.

  - [bdubbs] - vim-8.2.0129.로 업데이트. #4500.
  - [bdubbs] - gmp-6.2.0.로 업데이트. #4577 수정됨.
  - [bdubbs] - sed-4.8.로 업데이트. #4576 수정됨.
  - [bdubbs] - bc-2.5.1.로 업데이트. #4575 수정됨.
  - [bdubbs] - linux-5.4.13.로 업데이트. #4572 수정됨.
- 2020-01-16
  - [pierre] - libcap-2.31.로 업데이트. #4574 수정됨.
- 2020-01-13
  - [bdubbs] - zstd 라이브러리 설치 위치 수정
- 2020-01-12
  - [bdubbs] - zstd-1.4.4 추가됨.
- 2020-01-09
  - [bdubbs] - meson-0.53.0.로 업데이트. #4571 수정됨.
  - [bdubbs] - e2fsprogs-1.45.5.로 업데이트. #4570 수정됨.
  - [bdubbs] - grep-3.4.로 업데이트. #4568 수정됨.
  - [bdubbs] - libpipeline-1.5.2.로 업데이트. #4567 수정됨.
  - [bdubbs] - linux-5.4.8.로 업데이트. #4566 수정됨.
  - [pierre] - BLFS의 일부 패키지에 필요한 /etc/os-release sysV 버전에 추가됨.
- 2020-01-06
  - [pierre] - libcap-2.30.로 업데이트. #4569 수정됨.
- 2020-01-04
  - [pierre] - libcap-2.29의 다양한 이슈 수정. 의존성 업데이트.
- 2020-01-01
  - [bdubbs] - libcap-2.29.로 업데이트. #4564 수정됨.
- 2019-12-22
  - [pierre] - python3-3.8.1.로 업데이트. #4564 수정됨.
  - [pierre] - file-5.38.로 업데이트. #4563 수정됨.
  - [pierre] - linux-5.4.6.로 업데이트. #4562 수정됨.
  - [pierre] - vim-8.2.0024.로 업데이트. #4500의 일부.
- 2019-12-12
  - [bdubbs] - libcap-2.28.로 업데이트. #4559 수정됨.
  - [bdubbs] - bison-3.5.로 업데이트. #4561 수정됨.
- 2019-12-10
  - [renodr] - 의존 패키지에서 문제를 일으킨 meson의 회귀 문제 수정.
- 2019-12-05
  - [renodr] - bc-2.4.0.로 업데이트. #4556 수정됨.

  - [renodr] - shadow-4.8.로 업데이트. #4557 수정됨.
  - [renodr] - linux-5.4.2.로 업데이트. #4558 수정됨.
- 2019-12-01
  - [bdubbs] - Bash를 위한 업스트림 수정 패치 추가. LFS 유저용으로 수정할 때 발생할 수 있는 문제점에 대한 메모 추가.
  - [bdubbs] - vim-8.1.2361.로 업데이트. #4500 업데이트됨.
  - [bdubbs] - meson-0.52.1.로 업데이트. #4555 수정됨.
  - [bdubbs] - elfutils-0.178.로 업데이트. #4553 수정됨.
  - [bdubbs] - iproute2-5.4.0.로 업데이트. #4551 수정됨.
  - [bdubbs] - libffi-3.3.로 업데이트. #4550 수정됨.
  - [bdubbs] - tcl-8.6.10.로 업데이트. #4549 수정됨.
  - [bdubbs] - man-pages-5.04.로 업데이트. #4548 수정됨.
  - [bdubbs] - perl-5.30.1.로 업데이트. #4547 수정됨.
  - [bdubbs] - linux-5.4.1.로 업데이트. #4546 수정됨.
  - [bdubbs] - bc-2.3.2.로 업데이트. #4545 수정됨.
- 2019-11-08
  - [renodr] - #4544 수정됨.
- 2019-10-31
  - [dj] - lfs-bootscripts-20191031.로 업데이트.
- 2019-10-25
  - [dj] - lfs-bootscripts-20191025.로 업데이트.
- 2019-11-01
  - [bdubbs] - linux-5.3.8.로 업데이트. #4539 수정됨.
  - [bdubbs] - bc-2.2.0.로 업데이트. #4543 수정됨.
  - [bdubbs] - check-0.13.0.로 업데이트. #4540 수정됨.
  - [bdubbs] - eudev-3.2.9.로 업데이트. #4542 수정됨.
  - [bdubbs] - man-db-2.9.0.로 업데이트. #4541 수정됨.
- 2019-10-17
  - [bdubbs] - attr과 acl을 shadow 패키지 앞으로 이동.
  - [bdubbs] - linux-5.3.6.로 업데이트. #4534 수정됨.
  - [bdubbs] - man-pages-5.03.로 업데이트. #4536 수정됨.
  - [bdubbs] - meson-0.52.0.로 업데이트. #4535 수정됨.
  - [bdubbs] - Python-3.8.0.로 업데이트. #4538 수정됨.
  - [bdubbs] - binutils-2.33.1.로 업데이트. #4537 수정됨.
- 2019-09-29
  - [bdubbs] - texinfo-6.7.로 업데이트. #4529 수정됨.

- [bdubbs] - e2fsprogs-1.45.4.로 업데이트. #4530 수정됨.
- [bdubbs] - XML-Parser-2.46.로 업데이트. #4531 수정됨.
- [bdubbs] - expat-2.2.9.로 업데이트. #4532 수정됨.
- [bdubbs] - iproute2-5.3.0.로 업데이트. #4533 수정됨.

- 2019-09-24
  - [pierre] - linux-5.3.1.로 업데이트. #4528 수정됨.
- 2019-09-14
  - [bdubbs] - expat-2.2.8.로 업데이트. #4527 수정됨.
  - [bdubbs] - bison-3.4.2.로 업데이트. #4526 수정됨.
  - [bdubbs] - linux-5.2.14.로 업데이트. #4522 수정됨.
  - [bdubbs] - openssl-1.1.1d.로 업데이트. #4523 수정됨.
  - [bdubbs] - sysvinit-2.96.로 업데이트. #4524 수정됨.
  - [bdubbs] - tzdata-2019c.로 업데이트. #4525 수정됨.
- 2019-09-02
  - [dj] - lfs-bootscripts-20190908.로 업데이트.
- 2019-09-02
  - [bdubbs] - linux-5.2.11.로 업데이트. #4517 수정됨.
  - [bdubbs] - man-db-2.8.7.로 업데이트. #4518 수정됨.
  - [bdubbs] - meson-0.51.2.로 업데이트. #4519 수정됨.
  - [bdubbs] - findutils-4.7.0.로 업데이트. #4520 수정됨.
  - [dj] - LFS-Bootscripts-20190902 - 부트스크립트의 LSB 종속성 정보를 수정하고 새 LSB-Tools 패키지에 대한 표준 업데이트 페이지를 수정
- 2019-09-01
  - [bdubbs] - LFS-9.0 출간됨.

1.4. 도움이 될만한 곳

1.4.1. FAQ

LFS 시스템을 빌드하는 동안 오류가 발생하거나, 질문이 있거나, 책에 오타가 있다면, 먼저 http://www.linuxfromscratch.org/faq/에 있는 FAQ(자주 묻는 질문)를 참고하라.

1.4.2. 메일링 리스트

linuxfromscratch.org 서버는 LFS 프로젝트의 개발에 사용되는 많은 메일링 리스트를 가지고 있다. 이 리스트에는 development와 support, 그 외 리스트들이 있다. FAQ로 문제가 해결되지 않으면 다음 단계는 http://www.linuxfromscratch.org/search.html에서 메일 목록을 검색하는 것이다.

다양한 리스트, 구독 방법, 보관 위치 및 추가 정보에 대한 자세한 내용은 http://www.linuxfromscratch.org/mail.html를 참고하라.

1.4.3. IRC

LFS 커뮤니티의 몇몇 구성원들은 인터넷 릴레이 채팅(IRC)에서 도움을 제공하고 있다. 이 곳에 질문하기 전에 LFS FAQ 또는 메일링 리스트 보관함에서 질문에 대한 답변이 아직 없는지 확인하라. IRC 네트워크는 irc.freenode.net에서 찾을 수 있다. 지원 채널의 이름은 #LFS-support 이다.

1.4.4. 미러 사이트

LFS 프로젝트는 웹사이트에 접속하고 필요한 패키지를 더 편리하게 다운로드 할 수 있도록 전세계적으로 많은 미러 사이트를 가지고 있다. 현재 미러 목록을 보려면 http://www.linuxfromscratch.org/mirrors.html의 LFS 웹사이트를 방문하라.

1.4.5. 문의 정보

모든 질문과 건의는 LFS 메일링 리스트 중 하나로 전달하라. (위 참고)

1.5. 도움말

이 책을 읽는 동안 문제나 질문이 생기면 http://www.linuxfromscratch.org/faq/#generalfaq의 FAQ 페이지를 확인하라. 많은 질문들은 그 곳에서 해결된다. 그 곳에서 질문에 대한 답변을 찾지 못하면 문제의 원인을 파악하라. 다음 링크는 문제 해결을 위한 몇 가지 안내를 제공할 것이다: http://www.linuxfromscratch.org/hints/downloads/files/errors.txt.

FAQ에서 당신의 질문에 대한 답을 찾지 못했다면, http://www.linuxfromscratch.org/search.html에서 메일링 리스트를 찾아보라.

메일링 리스트와 IRC를 통해 도움을 주고자 하는 훌륭한 LFS 커뮤니티도 있다 (이 책의 1.4절. “도움이 될만한 곳” 참고). 그러나 우리는 매일 여러 질문들을 받고 있고, 그 중 많은 질문들은 FAQ나 메일링 리스트를 검색하면 쉽게 해결될 수 있는 질문들이다. 그래서 우리가 가능한 최선의 해결책을 주기 위해선 먼저 당신 스스로 조사를 해야 한다. 이를 통해 보다 특수한 지원 요청에 집중할 수 있게 된다. 검색 결과 해결 방법이 없을 경우 질문 내용에 모든 관련 정보(아래 참고)를 포함하라.

1.5.1. 언급해야 할 사항

겪고있는 문제에 대한 간단한 설명 외에도, 도움을 요청하는 데 반드시 포함시켜야 할 필수 사항은 다음과 같다:

- 보고 있는 책의 버전 (지금의 경우 9.1)
- LFS 구축에 사용 중인 호스트 배포판과 버전
- Host System Requirements 스크립트의 결과
- 문제가 발생한 패키지나 섹션
- 정확한 오류 메시지 또는 증상
- 책의 절차를 정확하게 따랐는지의 여부

참고

이 책의 절차를 정확히 따르지 않았다고 해서 우리가 당신을 돕지 않겠다는 것은 아니다. 결국 LFS는 개인의 취향 문제다. 정해진 절차의 변경에 대해 솔직하게 말하는 것이 우리가 당신의 문제에 대한 원인을 찾고 해결하는데 도움이 된다.

1.5.2. 스크립트 Configure 문제

configure 스크립트를 실행하는 중 문제가 생겼다면 config.log 파일을 확인해보라. 이 파일에는 configure 중에 화면에 출력되지 않은 오류가 포함될 수 있다. 도움을 요청해야 할 경우 관련된 라인을 포함하라.

1.5.3. 컴파일 문제

화면 출력이나 다양한 파일의 내용 모두 컴파일 문제의 원인을 파악하는데 유용하다. configure 스크립트와 make 실행이 도움이 될 수 있다. 전체 출력을 포함할 필요는 없지만, 관련된 내용을 충분히 포함하라. 아래는 화면에 출력된 make의 결과 중 포함할 정보 유형의 예시이다:

```
gcc -DALIASPATH=₩"/mnt/lfs/usr/share/locale:.₩"
-DLOCALEDIR=₩"/mnt/lfs/usr/share/locale₩"
-DLIBDIR=₩"/mnt/lfs/usr/lib₩"
-DINCLUDEDIR=₩"/mnt/lfs/usr/include₩" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc -g -O2 -static -o make ar.o arscan.o commands.o dir.o
expand.o file.o function.o getopt.o implicit.o job.o main.o
misc.o read.o remake.o rule.o signame.o variable.o vpath.o
default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference
to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

이 경우, 많은 사람들이 단지 아래 부분만을 언급한다:

```
make [2]: *** [make] Error 1
```

이것은 단지 무엇이 잘못됐는지가 아니라 잘못되었다는 점만 이야기할 뿐이기 때문에 문제를 제대로 진단하기에 충분한 정보가 아니다. 위의 예와 같은 전체 섹션은 실행된 명령과 관련 오류 메시지가 포함되기 때문에 같이 언급해야 할 사항이다.

인터넷상의 도움을 요청하는 것에 관한 훌륭한 기사는 http://catb.org/~esr/faqs/smart-questions.html에서 온라인으로 볼 수 있다. 필요한 도움을 받을 수 있는 가능성을 높이려면 이 문서의 힌트를 읽고 따르라.

부 II. 빌드를 위한 준비

2장. 호스트 시스템 준비

2.1. 도입

이 장에서는 LFS 구축에 필요한 호스트 도구들을 확인하고 필요한 경우 설치한다. 그러면 LFS 시스템을 호스팅할 파티션이 준비된다. 우리는 파티션 자체를 만들고, 그 위에 파일 시스템을 만들어 마운트할 것이다.

2.2. 호스트 시스템 요구사항

호스트 시스템에는 버전이 다음보다 크거나 같은 소프트웨어가 있어야 한다. 대부분의 현대 리눅스 배포판에서는 문제가 되지 않는다. 또한 많은 배포판에서 소프트웨어 헤더를 종종 “<패키지-이름>-devel”나 “<패키지-이름>-dev” 형식으로 별도의 패키지에 표시한다. 배포판에서 그렇게 제공하는 경우 반드시 설치한다.

나열된 소프트웨어 패키지 버전의 이전 버전도 작동할 수는 있지만, 테스트되지는 않았다.

- Bash-3.2 (/bin/sh 가 Bash로 심볼릭 또는 하드링크 되어야 한다)
- Binutils-2.25 (2.34 이후의 버전은 테스트되지 않았기 때문에 권장되지 않음)
- Bison-2.7 (/usr/bin/yacc가 bison이나 bison을 실행하는 작은 스크립트에 링크되어야 함)
- Bzip2-1.0.4
- Coreutils-6.9
- Diffutils-2.8.1
- Findutils-4.2.31
- Gawk-4.0.1 (/usr/bin/awk가 gawk에 링크되어야 함)
- GCC-6.2 C++컴파일러인 g++를 포함 (9.2.0 이후의 버전은 테스트되지 않았기 때문에 권장되지 않음)
- Glibc-2.11 (2.31 이후의 버전은 테스트되지 않았기 때문에 권장되지 않음)
- Grep-2.5.1a
- Gzip-1.3.12
- Linux Kernel-3.2

  이 커널 버전이 필요한 이유는 개발자의 권장 사항에 따라 6장에서 glibc를 빌드할 때 그 버전을 지정하기 때문이다. 또 udev에도 필요하다.

  호스트 커널이 3.2 이전인 경우 커널을 보다 높은 버전으로 교체해야 한다. 이것을 할 수 있는 방법은 두 가지가 있다. 먼저 리눅스 벤더가 3.2 이상의 커널 패키지를 제공하는지 확인하라. 만약 그렇다면, 그걸 설치 하라. 벤더가 수용 가능한 커널 패키지를 제공하지 않거나 당신이 설치하고 싶지 않으면 직접 커널을 컴파일할 수도 있다. 커널을 컴파일 하고 부팅 로더를 구성하는 방법(호스트에서 GRUB를 사용하는 것으로 간주)은 8장에 수록되어 있다.
- M4-1.4.10
- Make-4.0
- Patch-2.5.4
- Perl-5.8.8
- Python-3.4
- Sed-4.1.5
- Tar-1.22
- Texinfo-4.7
- Xz-5.0.0

중요

위에 언급된 심볼링크는 이 책에 포함된 지침을 따라 LFS 시스템을 구축하기 위해 필요하다. dash나 mawk같은 다른 소프트웨어를 가리키는 Symlink도 작동하겠지만 LFS 개발팀에서 테스트하거나 다뤄지지는 않았으며, 책의 지침을 벗어나야 하거나 일부 패키지에 추가 패치가 필요할 수 있다.

호스트 시스템에 적절한 버전이 모두 있는지, 프로그램을 컴파일할 수 있는지 확인하려면 다음을 실행하라.:

```
cat > version-check.sh << "EOF"
#!/bin/bash
# Simple script to list version numbers of critical development tools
export LC_ALL=C
bash --version | head -n1 | cut -d" " -f2-4
MYSH=$(readlink -f /bin/sh)
echo "/bin/sh -> $MYSH"
echo $MYSH | grep -q bash || echo "ERROR: /bin/sh does not point to bash"
unset MYSH

echo -n "Binutils: "; ld --version | head -n1 | cut -d" " -f3-
bison --version | head -n1

if [ -h /usr/bin/yacc ]; then
  echo "/usr/bin/yacc -> `readlink -f /usr/bin/yacc`";
elif [ -x /usr/bin/yacc ]; then
  echo yacc is `/usr/bin/yacc --version | head -n1`
else
  echo "yacc not found"
fi

bzip2 --version 2>&1 < /dev/null | head -n1 | cut -d" " -f1,6-
echo -n "Coreutils: "; chown --version | head -n1 | cut -d")" -f2
diff --version | head -n1
find --version | head -n1
gawk --version | head -n1

if [ -h /usr/bin/awk ]; then
  echo "/usr/bin/awk -> `readlink -f /usr/bin/awk`";
elif [ -x /usr/bin/awk ]; then
  echo awk is `/usr/bin/awk --version | head -n1`
else
  echo "awk not found"
fi
```

```
gcc --version | head -n1
g++ --version | head -n1
ldd --version | head -n1 | cut -d" " -f2-  # glibc version
grep --version | head -n1
gzip --version | head -n1
cat /proc/version
m4 --version | head -n1
make --version | head -n1
patch --version | head -n1
echo Perl `perl -V:version`
python3 --version
sed --version | head -n1
tar --version | head -n1
makeinfo --version | head -n1  # texinfo version
xz --version | head -n1
```

```
echo 'int main(){}' > dummy.c && g++ -o dummy dummy.c
if [ -x dummy ]
  then echo "g++ compilation OK";
  else echo "g++ compilation failed"; fi
rm -f dummy.c dummy
EOF

bash version-check.sh
```

2.3. LFS 빌드 단계

LFS는 한 세션에 구축되도록 설계되었다. 즉, 절차 도중에는 시스템이 종료되지 않을 것으로 가정한다. 그렇다고 시스템이 앉은 자리에서 한번에 모두 설치되어야 한다는 뜻은 아니다. 중요한 것은 다른 시점에 LFS를 재개할 경우, 재부팅 후 특정 절차를 다시 수행해야 한다는 점이다.

2.3.1. 1-4장

이 장들은 호스트 시스템에서 수행된다. 중단 후 재개할 경우 다음을 주의하라:

- 2장 4절 이후 root 사용자로 수행되는 절차는 root 사용자에 대해 LFS 환경 변수를 설정해야 한다.

2.3.2. 5장

- /mnt/lfs 파티션이 반드시 마운트되어야 한다.
- 5장의 모든 지침은 반드시 lfs 유저에 의해 수행되어야 한다. su - lfs는 5장의 어떤 작업보다도 먼저 실행되어야 한다.
- 5.3절. “일반적인 컴파일 절차”의 내용은 매우 중요하다. 패키지를 설치할 때 확신이 들지 않는다면, 이전에 압축을 푼 tarball을 확실히 제거하고 패키지 파일들을 다시 추출한 후 해당 절의 모든 절차를 완료하라.

2.3.3. 6-8장

- /mnt/lfs 파티션이 반드시 마운트되어야 한다.
- chroot로 진입할 때는 반드시 root에 대해 LFS 환경 변수를 설정해야 한다. LFS 변수는 달리 사용되지 않는다.
- 가상 파일 시스템을 반드시 마운트해야 한다. 이는 호스트 가상 터미널로 변경하고 root로 6.2.2절. “/dev 마운팅과 설정”과 6.2.3절. “가상 커널 파일 시스템 마운팅”의 명령을 실행하여 chroot로 진입하기 전 또는 후에 수행할 수 있다.

2.4. 새 파티션 만들기

대부분의 다른 운영 체제와 마찬가지로 LFS도 전용 파티션에 설치된다. LFS 시스템을 구축하기 위한 권장 방식은 사용 가능한 빈 파티션을 사용하거나, 파티션이 지정되지 않은 충분한 여유 공간이 있으면 새로 생성하는 것이다.

최소한의 시스템은 약 10기가바이트(GB)의 파티션을 필요로 한다. 이것은 모든 소스 tarball을 저장하고 패키지를 컴파일하기에 충분하다. 그러나 LFS 시스템을 주 리눅스 시스템으로 쓴다면 추가 공간을 필요로 하는 소프트웨어를 더 설치해야할 것이다. 30GB의 파티션이 추가적으로 소프트웨어를 설치하기 위해 적당한 크기이다. LFS 시스템 자체는 이렇게 많은 공간을 차지하지 않는다. 이 요구 용량의 상당 부분은 LFS가 완료된 후 기능을 추가하기 위한 것뿐만 아니라 충분한 여유 임시 공간을 제공하려는 것이다. 또한 패키지를 컴파일하려면 많은 디스크 공간이 필요할 수도 있으며 패키지를 설치한 후 회수할 수 있다.

컴파일 프로세스에 사용할 수 있는 메모리(RAM)가 항상 충분한 것은 아니기 때문에 작은 디스크 파티션을 swap 공간으로 사용하는 것이 좋다. 이것은 거의 사용되지 않는 데이터를 저장하고 활성 프로세스에 더 많은 메모리를 사용할 수 있도록 커널에서 사용된다. LFS 시스템의 swap 파티션은 호스트 시스템에서 사용하는 파티션과 같을 수 있으며, 이 경우 다른 파티션을 만들 필요가 없다.

cfdisk 또는 fdisk와 같은 디스크 파티셔닝 프로그램과 함께, 파티션이 만들어질 하드 디스크를 지정하는 명령 줄 옵션을 사용한다—예를 들어 주 디스크 드라이브는 /dev/sda. 필요한 경우 리눅스 기본(native) 파티션 및 swap 파티션을 생성하라. 프로그램 사용법을 아직 모른다면 cfdisk(8) 또는 fdisk(8)를 참고하라.

참고

숙련된 사용자라면 다른 파티션 분할 방식도 가능하다. 이 새 LFS 시스템은 소프트웨어 RAID 어레이 또는 LVM 논리적 볼륨 상에 설치할 수도 있다. 그러나 이러한 선택지 중 일부는 고급 주제인 initramfs를 필요로 한다. 이러한 파티션 분할 방법들은 LFS 처음 사용자에게는 권장되지 않는다.

새 파티션의 명칭(가령, sda5)을 기억하라. 이 책에서는 이것을 LFS 파티션 이라고 칭할 것이다. swap 파티션의 명칭도 기억하라. 이들은 나중에 /etc/fstab 파일에 필요할 것이다.

2.4.1. 다른 파티션에 관해

시스템 파티션 분배에 대해 조언을 구하는 글은 LFS 메일링 리스트에 드물지 않게 올라온다. 이것은 매우 주관적인 주제이다. 대부분의 배포판에서 기본값은 하나의 작은 swap 파티션을 제외하고 전체 드라이브를 사용하는 것이다. 이것은 여러 가지 이유에서 LFS에 최적이 아니다. 유연성을 줄이고 여러 배포판 또는 LFS 시스템 내의 데이터 공유를 더욱 어렵게 하며, 백업 시간이 더 많이 소요되고 파일 시스템 구조의 비효율적인 할당에 따라 디스크 공간을 낭비할 수 있다.

2.4.1.1. Root 파티션

10 기가바이트의 Root LFS 파티션(/root 디렉토리와 혼동하지 말라)은 대부분의 시스템에 적당한 절충안이다. LFS와 대부분의 BLFS를 구축할 수 있는 충분한 공간을 제공하면서도, 여러 개의 파티션을 만들어서 쓰기에도 쉬울만큼 적당히 작다.

2.4.1.2. Swap 파티션

대부분의 배포판은 자동으로 swap 파티션을 생성한다. 일반적으로 swap 파티션의 권장 크기는 물리적 RAM의 약 2배지만, 보통 이만큼까지 필요하지는 않다. 디스크 공간이 제한적일 경우 swap 파티션을 2GB로 유지하고 디스크가 스왑되는 정도를 관찰하라.

스와핑은 결코 좋지 않다. 일반적으로 시스템이 디스크 작업 명령을 듣고 어떻게 반응하는지 관찰하기만 하면 시스템이 스와핑되는지 알 수 있다. 5기가바이트짜리 파일을 편집하려고 하는 등의 불합리한 명령으로 확인하는 것이다. 스와핑 이 자주 발생하면 시스템에 더 많은 RAM을 구입해 장착하는 것이 가장 좋은 해결책이다.

2.4.1.3. Grub Bios 파티션

부트 디스크가 GUID 파티션 테이블(GPT)로 만들어진 경우, 약 1MB의 작은 파티션이 아직 존재하지 않는다면 반드시 생성해야 한다. 이 파티션은 포맷하지 않아도 되지만, 부트 로더 설치 중에 GRUB이 이용할 수 있어야 한다. 이 파티션은 보통 fdisk를 사용하면 'BIOS Boot' 라벨이 붙고 gdisk를 사용하면 EF02 코드를 갖는다.

참고

Grub Bios 파티션은 BIOS가 시스템을 부팅하는 데 사용하는 드라이브에 있어야 한다. 이것은 LFS root 파티션이 있는 드라이브와 반드시 같을 필요는 없다. 시스템의 디스크들은 각기 다른 파티션 테이블 유형을 사용할 수 있다. 이 파티션에 대한 요구 사항은 부트 디스크의 파티션 테이블 유형에 따라서만 달라진다.

2.4.1.4. 편의성 파티션

필요하지는 않지만 디스크 레이아웃을 정할 때 고려되는 몇가지 다른 파티션이 있다. 다음 목록은 꼭 따르지 않아도 좋은, 일종의 가이드이다.

- /boot - 적극 권장. 이 파티션을 커널과 다른 부팅 정보들을 저장하는데 사용하라. 용량이 큰 디스크들의 잠재적인 부팅 문제를 최소화하려면, 이 파티션을 첫 번째 디스크의 첫 번째 물리적 파티션으로 생성하라. 파티션 크기는 100MB가 적당하다.
- /home - 적극 권장. 여러 배포판들과 LFS 시스템 사이에서 당신의 홈 디렉토리와 사용자 정의 설정들을 공유하라. 크기는 일반적으로 상당히 커야하고 가용한 디스크 공간에 따라 다르다.
- /usr - 별도의 /usr 파티션은 보통 작은 클라이언트에 서버를 제공하거나 디스크가 없는 워크스테이션일 경우에 쓰인다. 일반적인 LFS에는 필요하지 않다. 5기가바이트의 용량이면 대부분의 설치 작업에 충분하다.
- /opt - 이 디렉토리는 /usr 계층에 파일을 포함시키지 않고 Gnome이나 KDE와 같은 대형 패키지를 여러번 설치하는 BLFS에 가장 유용하다. 사용하려면 일반적으로 5~10기가바이트가 적당하다.
- /tmp - 별도의 /tmp 디렉토리는 흔치 않지만 작은 클라이언트를 구성할 때 유용하다. 이 파티션은 보통 수 기가바이트를 초과할 일이 없다.
- /usr/src - 이 파티션은 BLFS 소스 파일을 저장하고 LFS 빌드들과 공유할 수 있는 위치로써 매우 유용하다. 또한 BLFS 패키지를 빌드하는 위치로도 사용할 수 있다. 30~50기가바이트의 적당히 큰 파티션이면 공간이 넉넉할 것이다.

부팅 시 자동으로 마운트할 각 파티션을 /etc/fstab에 지정해야 한다. 파티션 지정 방법에 대한 자세한 내용은 8.2절. “/etc/fstab 파일 생성”에서 설명한다.

2.5. 파티션에 파일 시스템 만들기

빈 파티션이 준비되었으므로 이제 파일 시스템을 만들 수 있다. LFS는 리눅스 커널에서 인식되는 모든 파일 시스템을 사용할 수 있지만 가장 일반적인 타입은 ext3와 ext4이다. 파일 시스템의 선택은 복잡한 사항이며 파일들의 특성과 파티션 크기에 따라 달라질 수 있다. 예를 들어:

ext2
: 이 파일 시스템은 /boot같은 자주 업데이트되지 않는 작은 파티션에 적합하다.

ext3
: ext2의 업그레이드 버전이며 비정상적인 시스템 종료에서 파티션의 상태를 복구하는데 도움을 주는 저널 기능이 포함되어 있다. 범용 파일 시스템으로 널리 쓰인다.

ext4
: 파티션 타입 중 ext 파일 시스템군의 최신 버전이다. 나노초단위 타임스탬프, 초대형 파일 (16TB) 생성 및 사용, 속도 향상 등 몇가지 새로운 기능들을 제공한다.

FAT32, NTFS, ReiserFS, JFS, XFS와 같은 다른 파일 시스템들도 특수한 용도로 쓰인다. 이러한 파일 시스템에 대한 자세한 내용은 http://en.wikipedia.org/wiki/Comparison_of_file_systems에서 확인하라.

LFS는 root 파일 시스템(/)이 ext4 타입이라고 간주한다. LFS 파티션에 ext4 파일 시스템을 생성하려면 다음을 실행하라:

```
mkfs -v -t ext4 /dev/<xxx>
```

만약 기존의 swap 파티션을 사용한다면, 포맷할 필요가 없다. 새로운 swap 파티션을 만들었다면, 다음 명령으로 초기화해야 한다:

```
mkswap /dev/<yyy>
```

swap 파티션의 이름으로 <yyy> 를 대체하라.

2.6. $LFS 변수 설정

이 책 전반에 걸쳐 환경변수 LFS를 여러 차례 사용할 것이다. 이 변수가 LFS 빌드 과정 전체에 걸쳐 항상 정의되어 있는지 확인하라. 이것은 당신의 LFS 시스템을 구축할 디렉토리 이름으로 설정되어야 한다. - 우리는 예로 /mnt/lfs를 사용할 것이지만, 디렉토리 선택은 당신에게 달려 있다. 별도의 파티션에 LFS를 구축하려는 경우 그 파티션을 마운트할 지점의 디렉토리 이름으로 변수를 설정하라. 디렉토리 위치를 선택하고 다음 명령으로 변수를 설정하라:

```
export LFS=/mnt/lfs
```

이 변수를 설정하면 mkdir -v $LFS/tools와 같은 명령을 문자 그대로 입력할 수 있다는 이점이 있다. 셸은 명령 줄을 처리할 때 “$LFS”를 자동으로 “/mnt/lfs”로 (또는 따로 설정된 값으로) 바꾸어 준다.

경고

현재 작업 환경을 종료하고 다시 들어갈 때마다(가령 root나 다른 유저로의 로그인을 위해 su를 입력할 때) LFS가 설정되어 있는지 확인하는 것을 잊지 말라. LFS 변수가 올바르게 설정되었는지 다음을 통해 확인하라:

```
echo $LFS
```

LFS 시스템의 빌드 위치에 대한 주소가 출력되는지 확인하라. 책의 예시를 따랐을 경우 이 주소는 /mnt/lfs이다. 출력이 올바르지 않다면 이 페이지의 앞서 언급한 명령을 사용해 $LFS를 올바른 디렉토리 이름으로 설정하라.

참고

LFS 변수가 항상 설정되도록 하는 한 가지 방법은 개인 홈 디렉토리와 /root/.bash_profile 모두에서 .bash_profile 파일에 위의 export 명령을 입력하는 것이다. 또한 /root/.bash_profile 파일이 로그인 프로세스 중 인식되게 하기 위해 LFS 변수가 필요한 모든 사용자에 대해 /etc/passwd 파일에 지정된 셸을 Bash로 만들어야 한다.

호스트 시스템에 로그인하는 방식도 고려해야 한다. 일반적으로 그래픽 디스플레이 관리자를 통해 로그인하는 경우 가상 터미널이 시작될 때 사용자의 .bash_profile은 사용되지 않는다. 이 경우 export 명령을 사용자 및 루트의 .bashrc 파일에 추가하라. 또한 일부 배포판에는 비대화형(non-interactive) Bash 호출에선 .bashrc를 불러오지 않는 경우가 있다. 비대화형 테스트를 수행하기 전에 export 명령을 추가하라.

2.7. 새 파티션 마운팅

파일 시스템이 만들어졌으니, 이제 파티션에 접근할 수 있게 해야 한다. 이를 위해 파티션을 선택한 마운트 지점에 마운트해야 한다. 본서의 목적상, 파일 시스템은 앞의 절에서 기술한 바와 같이 LFS 환경변수에 지정된 디렉토리 아래에 마운트되는 것으로 가정한다.

다음과 같이 마운트 지점을 생성하고 LFS 파일 시스템을 마운트한다:

```
mkdir -pv $LFS
mount -v -t ext4 /dev/<xxx> $LFS
```

LFS 파티션의 이름으로 <xxx>를 대체하라.

LFS에 여러 파티션들을 사용한다면 (가령, 하나는 /이고 다른 하나는 /usr라면), 다음과 같이 마운트한다:

```
mkdir -pv $LFS
mount -v -t ext4 /dev/<xxx> $LFS
mkdir -v $LFS/usr
mount -v -t ext4 /dev/<yyy> $LFS/usr
```

각 파티션의 이름으로 <xxx>와 <yyy>를 대체하라.

이 새 파티션에 너무 제한적인 사용 권한(예를 들어 nosuid 또는 nodev 옵션) 으로 마운트되어 있지는 않은지 확인하라. 마운트된 LFS 파티션에 대해 설정된 옵션을 확인하려면 매개 변수 없이 mount 명령을 실행하라. nosuid나 또는 nodev가 설정되어 있으면 파티션을 다시 마운트해야한다.

주의

위의 지침은 LFS 절차 중에 컴퓨터를 다시 시작하지 않을 것이라고 가정한다. 시스템을 종료할 경우 빌드 절차를 재개할 때마다 LFS 파티션을 다시 마운트하거나 부팅할 때 자동으로 다시 마운트하도록 호스트 시스템의 /etc/fstab file 파일을 수정해야 한다. 예를 들면:

```
/dev/<xxx>  /mnt/lfs ext4    defaults      1     1
```

추가적으로 파티션을 더 사용한다면 해당 파티션도 추가하라.

swap 파티션을 사용한다면, 다음과 같이 swapon 명령으로 활성화되어 있는지 확인하라:

```
/sbin/swapon -v /dev/<zzz>
```

swap 파티션의 이름으로 <zzz> 를 대체하라.

이제 작업할 공간이 만들어졌으니, 패키지들을 다운로드할 차례이다.

3장. 패키지와 패치들

3.1. 도입

이 장에는 기본적인 리눅스 시스템을 구축하기 위해 다운로드해야 하는 패키지 목록이 포함되어 있다. 명시된 버전 숫자는 소프트웨어가 작동하는 것으로 알려진 버전이며, 이 책은 그것들을 기반으로 작성되었다. 우리는 한 버전에서의 빌드 명령이 새로운 버전에서는 작동하지 않을 수 있기 때문에 새로운 버전을 사용하지 않을 것을 강력히 권고한다. 또한 최신 패키지 버전에는 해결이 필요한 문제가 있을 수 있다. 그 해결책들은 이 책의 개발 버전에서 개발되고 안정화될 것이다.

다운로드 링크가 항상 사용 가능하지는 않을 수도 있다. 이 책이 출간된 이후 다운로드 위치가 바뀌었다면 구글(http://www.google.com/)은 대부분의 패키지에 유용한 검색 엔진을 제공한다. 만약 찾지 못하겠다면 http://www.linuxfromscratch.org/lfs/packages.html#packages에서 설명하는 다른 다운로드 방법 중 하나를 시도해보라.

다운로드된 패키지와 패치는 전체 빌드에서 편리하게 사용할 수 있는 곳에 저장되어야 할 것이다. 소스의 압축을 풀고 그것을 빌드하기 위해서도 작업 디렉토리가 필요하다. $LFS/sources는 tarball과 패치를 저장하는 장소와 작업 디렉토리로 모두 사용될 수 있다. 이 디렉토리를 사용함으로써, 필요한 요소들은 LFS 파티션에 위치하게 될 것이고, 빌드 프로세스의 모든 단계에서 이용 가능할 것이다.

이 디렉토리를 만들려면 다운로드 섹션을 시작하기 전에 root 유저로 다음 명령을 실행하라:

```
mkdir -v $LFS/sources
```

이 디렉토리를 쓰기 가능(writable)하고 고정적(sticky)으로 만들어라. “고정적”이란 여러 명의 사용자가 디렉토리에 대한 쓰기 권한을 갖고 있더라도 파일 소유자만이 해당 디렉토리 내에서 파일을 삭제할 수 있다는 것을 의미한다. 다음 명령은 쓰기 및 고정 모드를 활성화한다:

```
chmod -v a+wt $LFS/sources
```

손쉽게 모든 패키지와 패치를 다운로드하는 방법은 wget-list를 wget에 입력하는 것이다. 예를 들면 다음과 같다:

```
wget --input-file=wget-list --continue --directory-prefix=$LFS/sources
```

추가로 LFS-7.0부터는 별도의 파일로 md5sums가 있어, 진행하기 전에 모든 패키지들이 올바른지 확인할 수 있다. 그 파일을 $LFS/sources에 넣고 다음을 실행하라:

```
pushd $LFS/sources
md5sum -c md5sums
popd
```

3.2. 모든 패키지들

다음 패키지들을 다운로드하라:

- Acl (2.2.53) - 513 KB:

홈페이지: https://savannah.nongnu.org/projects/acl
다운로드: http://download.savannah.gnu.org/releases/acl/acl-2.2.53.tar.gz
MD5 sum: 007aabf1dbb550bcddde52a244cd1070

- Attr (2.4.48) - 457 KB:
홈페이지: https://savannah.nongnu.org/projects/attr
다운로드: http://download.savannah.gnu.org/releases/attr/attr-2.4.48.tar.gz
MD5 sum: bc1e5cb5c96d99b24886f1f527d3bb3d
- Autoconf (2.69) - 1,186 KB:
홈페이지: http://www.gnu.org/software/autoconf/
다운로드: http://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar.xz
MD5 sum: 50f97f4159805e374639a73e2636f22e
- Automake (1.16.1) - 1,499 KB:
홈페이지: http://www.gnu.org/software/automake/
다운로드: http://ftp.gnu.org/gnu/automake/automake-1.16.1.tar.xz
MD5 sum: 53f38e7591fa57c3d2cee682be668e5b
- Bash (5.0) - 9,898 KB:
홈페이지: http://www.gnu.org/software/bash/
다운로드: http://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
MD5 sum: 2b44b47b905be16f45709648f671820b
- Bc (2.5.3) - 247 KB:
홈페이지: https://github.com/gavinhoward/bc
다운로드: https://github.com/gavinhoward/bc/archive/2.5.3/bc-2.5.3.tar.gz
MD5 sum: 6582c6fbbae943fbfb8fe14a34feab57
- Binutils (2.34) - 21,131 KB:
홈페이지: http://www.gnu.org/software/binutils/
다운로드: http://ftp.gnu.org/gnu/binutils/binutils-2.34.tar.xz
MD5 sum: 664ec3a2df7805ed3464639aaae332d6
- Bison (3.5.2) - 2,308 KB:
홈페이지: http://www.gnu.org/software/bison/
다운로드: http://ftp.gnu.org/gnu/bison/bison-3.5.2.tar.xz
MD5 sum: 49fc2cf23e31e697d5072835e1662a97
- Bzip2 (1.0.8) - 792 KB:
다운로드: https://www.sourceware.org/pub/bzip2/bzip2-1.0.8.tar.gz
MD5 sum: 67e051268d0c475ea773822f7500d0e5
- Check (0.14.0) - 753 KB:
홈페이지: https://libcheck.github.io/check
다운로드: https://github.com/libcheck/check/releases/download/0.14.0/check-0.14.0.tar.gz
MD5 sum: 270e82a445be6026040267a5e11cc94b
- Coreutils (8.31) - 5,284 KB:
홈페이지: http://www.gnu.org/software/coreutils/
다운로드: http://ftp.gnu.org/gnu/coreutils/coreutils-8.31.tar.xz
MD5 sum: 0009a224d8e288e8ec406ef0161f9293
- DejaGNU (1.6.2) - 514 KB:
홈페이지: http://www.gnu.org/software/dejagnu/
다운로드: http://ftp.gnu.org/gnu/dejagnu/dejagnu-1.6.2.tar.gz
MD5 sum: e1b07516533f351b3aba3423fafeffd6

- Diffutils (3.7) - 1,415 KB:
홈페이지: http://www.gnu.org/software/diffutils/
다운로드: http://ftp.gnu.org/gnu/diffutils/diffutils-3.7.tar.xz
MD5 sum: 4824adc0e95dbbf11dfbdfaad6a1e461

- E2fsprogs (1.45.5) - 7,753 KB:
홈페이지: http://e2fsprogs.sourceforge.net/
다운로드: https://downloads.sourceforge.net/project/e2fsprogs/e2fsprogs/v1.45.5/e2fsprogs-1.45.5.tar.gz
MD5 sum: 6d35428e4ce960cb7e875afe5849c0f3

- Elfutils (0.178) - 8,797 KB:
홈페이지: https://sourceware.org/ftp/elfutils/
다운로드: https://sourceware.org/ftp/elfutils/0.178/elfutils-0.178.tar.bz2
MD5 sum: 5480d0b7174446aba13a6adde107287f

- Eudev (3.2.9) - 1,914 KB:
다운로드: https://dev.gentoo.org/~blueness/eudev/eudev-3.2.9.tar.gz
MD5 sum: dedfb1964f6098fe9320de827957331f

- Expat (2.2.9) - 413 KB:
홈페이지: https://libexpat.github.io/
다운로드: https://prdownloads.sourceforge.net/expat/expat-2.2.9.tar.xz
MD5 sum: d2384fa607223447e713e1b9bd272376

- Expect (5.45.4) - 618 KB:
홈페이지: https://core.tcl.tk/expect/
다운로드: https://prdownloads.sourceforge.net/expect/expect5.45.4.tar.gz
MD5 sum: 00fce8de158422f5ccd2666512329bd2

- File (5.38) - 911 KB:
홈페이지: https://www.darwinsys.com/file/
다운로드: ftp://ftp.astron.com/pub/file/file-5.38.tar.gz
MD5 sum: 3217633ed09c7cd35ed8d04191675574

참고
File (5.38)은 더 이상 저 주소에서 구할 수 없을 수도 있다. 저 마스터 다운로드 주소의 사이트 관리자는 새 버전이 배포될 때 이전 버전을 삭제하는 경우가 있다. 정확한 버전을 구할 수 있는 다른 다운로드 주소는 이 곳에서도 찾을 수 있다: http://www.linuxfromscratch.org/lfs/download.html#ftp

- Findutils (4.7.0) - 1,851 KB:
홈페이지: http://www.gnu.org/software/findutils/
다운로드: http://ftp.gnu.org/gnu/findutils/findutils-4.7.0.tar.xz
MD5 sum: 731356dec4b1109b812fecfddfead6b2

- Flex (2.6.4) - 1,386 KB:
홈페이지: https://github.com/westes/flex
다운로드: https://github.com/westes/flex/releases/download/v2.6.4/flex-2.6.4.tar.gz
MD5 sum: 2882e3179748cc9f9c23ec593d6adc8d

- Gawk (5.0.1) - 3,063 KB:
홈페이지: http://www.gnu.org/software/gawk/
다운로드: http://ftp.gnu.org/gnu/gawk/gawk-5.0.1.tar.xz
MD5 sum: f9db3f6715207c6f13719713abc9c707

- GCC (9.2.0) - 68,953 KB:
홈페이지: https://gcc.gnu.org/
다운로드: http://ftp.gnu.org/gnu/gcc/gcc-9.2.0/gcc-9.2.0.tar.xz
MD5 sum: 3818ad8600447f05349098232c2ddc78

- GDBM (1.18.1) - 920 KB:
홈페이지: http://www.gnu.org/software/gdbm/
다운로드: http://ftp.gnu.org/gnu/gdbm/gdbm-1.18.1.tar.gz
MD5 sum: 988dc82182121c7570e0cb8b4fcd5415

- Gettext (0.20.1) - 9,128 KB:
홈페이지: http://www.gnu.org/software/gettext/
다운로드: http://ftp.gnu.org/gnu/gettext/gettext-0.20.1.tar.xz
MD5 sum: 9ed9e26ab613b668e0026222a9c23639

- Glibc (2.31) - 16,286 KB:
홈페이지: http://www.gnu.org/software/libc/
다운로드: http://ftp.gnu.org/gnu/glibc/glibc-2.31.tar.xz
MD5 sum: 78a720f17412f3c3282be5a6f3363ec6

- GMP (6.2.0) - 1,966 KB:
홈페이지: http://www.gnu.org/software/gmp/
다운로드: http://ftp.gnu.org/gnu/gmp/gmp-6.2.0.tar.xz
MD5 sum: a325e3f09e6d91e62101e59f9bda3ec1

- Gperf (3.1) - 1,188 KB:
홈페이지: http://www.gnu.org/software/gperf/
다운로드: http://ftp.gnu.org/gnu/gperf/gperf-3.1.tar.gz
MD5 sum: 9e251c0a618ad0824b51117d5d9db87e

- Grep (3.4) - 1,520 KB:
홈페이지: http://www.gnu.org/software/grep/
다운로드: http://ftp.gnu.org/gnu/grep/grep-3.4.tar.xz
MD5 sum: 111b117d22d6a7d049d6ae7505e9c4d2

- Groff (1.22.4) - 4,044 KB:
홈페이지: http://www.gnu.org/software/groff/
다운로드: http://ftp.gnu.org/gnu/groff/groff-1.22.4.tar.gz
MD5 sum: 08fb04335e2f5e73f23ea4c3adbf0c5f

- GRUB (2.04) - 6,245 KB:
홈페이지: http://www.gnu.org/software/grub/
다운로드: https://ftp.gnu.org/gnu/grub/grub-2.04.tar.xz
MD5 sum: 5aaca6713b47ca2456d8324a58755ac7

- Gzip (1.10) - 757 KB:
홈페이지: http://www.gnu.org/software/gzip/
다운로드: http://ftp.gnu.org/gnu/gzip/gzip-1.10.tar.xz
MD5 sum: 691b1221694c3394f1c537df4eee39d3
- Iana-Etc (2.30) - 201 KB:
홈페이지: http://freecode.com/projects/iana-etc
다운로드: http://anduin.linuxfromscratch.org/LFS/iana-etc-2.30.tar.bz2
MD5 sum: 3ba3afb1d1b261383d247f46cb135ee8
- Inetutils (1.9.4) - 1,333 KB:
홈페이지: http://www.gnu.org/software/inetutils/
다운로드: http://ftp.gnu.org/gnu/inetutils/inetutils-1.9.4.tar.xz
MD5 sum: 87fef1fa3f603aef11c41dcc097af75e
- Intltool (0.51.0) - 159 KB:
홈페이지: https://freedesktop.org/wiki/Software/intltool
다운로드: https://launchpad.net/intltool/trunk/0.51.0/+download/intltool-0.51.0.tar.gz
MD5 sum: 12e517cac2b57a0121cda351570f1e63
- IPRoute2 (5.5.0) - 731 KB:
홈페이지: https://www.kernel.org/pub/linux/utils/net/iproute2/
다운로드: https://www.kernel.org/pub/linux/utils/net/iproute2/iproute2-5.5.0.tar.xz
MD5 sum: ee8e2cdb416d4a8ef39525d39ab7c2d0
- Kbd (2.2.0) - 1,090 KB:
홈페이지: http://ftp.altlinux.org/pub/people/legion/kbd
다운로드: https://www.kernel.org/pub/linux/utils/kbd/kbd-2.2.0.tar.xz
MD5 sum: d1d7ae0b5fb875dc082731e09cd0c8bc
- Kmod (26) - 540 KB:
다운로드: https://www.kernel.org/pub/linux/utils/kernel/kmod/kmod-26.tar.xz
MD5 sum: 1129c243199bdd7db01b55a61aa19601
- Less (551) - 339 KB:
홈페이지: http://www.greenwoodsoftware.com/less/
다운로드: http://www.greenwoodsoftware.com/less/less-551.tar.gz
MD5 sum: 4ad4408b06d7a6626a055cb453f36819
- LFS-Bootscripts (20191031) - 32 KB:
다운로드: http://www.linuxfromscratch.org/lfs/downloads/9.1/lfs-bootscripts-20191031.tar.xz
MD5 sum: e9249541960df505e4dfac0c32369372
- Libcap (2.31) - 97 KB:
홈페이지: https://sites.google.com/site/fullycapable/
다운로드: https://www.kernel.org/pub/linux/libs/security/linux-privs/libcap2/libcap-2.31.tar.xz
MD5 sum: 52120c05dc797b01f5a7ae70f4335e96
- Libffi (3.3) - 1,275 KB:
홈페이지: https://sourceware.org/libffi/
다운로드: ftp://sourceware.org/pub/libffi/libffi-3.3.tar.gz
MD5 sum: 6313289e32f1d38a9df4770b014a2ca7

• Libpipeline (1.5.2) - 971 KB:
홈페이지: http://libpipeline.nongnu.org/
다운로드: http://download.savannah.gnu.org/releases/libpipeline/libpipeline-1.5.2.tar.gz
MD5 sum: 169de4cc1f6f7f7d430a5bed858b2fd3

• Libtool (2.4.6) - 951 KB:
홈페이지: http://www.gnu.org/software/libtool/
다운로드: http://ftp.gnu.org/gnu/libtool/libtool-2.4.6.tar.xz
MD5 sum: 1bfb9b923f2c1339b4d2ce1807064aa5

• Linux (5.5.3) - 108,112 KB:
홈페이지: https://www.kernel.org/
다운로드: https://www.kernel.org/pub/linux/kernel/v5.x/linux-5.5.3.tar.xz
MD5 sum: 3ea50025d8c679a327cf2fc225d81a46

참고
리눅스 커널은 보안 취약점의 발견으로 인해 비교적 자주 업데이트 된다. 정오표 페이지에 달리 명시되지 않는 한, 최신 5.5.x 커널 버전을 사용해야 한다.
리눅스 커널을 업데이트하기에 속도가 제한적이거나 대역폭 이용료가 비싼 사용자라면 패키지와 패치의 baseline 버전을 별도로 다운로드할 수 있다. 이렇게 하면 마이너 릴리스 내에서 후속 패치 업그레이드에 소요되는 시간이나 비용이 절약될 수 있다.

• M4 (1.4.18) - 1,180 KB:
홈페이지: http://www.gnu.org/software/m4/
다운로드: http://ftp.gnu.org/gnu/m4/m4-1.4.18.tar.xz
MD5 sum: 730bb15d96fffe47e148d1e09235af82

• Make (4.3) - 2,263 KB:
홈페이지: http://www.gnu.org/software/make/
다운로드: http://ftp.gnu.org/gnu/make/make-4.3.tar.gz
MD5 sum: fc7a67ea86ace13195b0bce683fd4469

• Man-DB (2.9.0) - 1,814 KB:
홈페이지: https://www.nongnu.org/man-db/
다운로드: http://download.savannah.gnu.org/releases/man-db/man-db-2.9.0.tar.xz
MD5 sum: 897576a19ecbef376a916485608cd790

• Man-pages (5.05) - 1,649 KB:
홈페이지: https://www.kernel.org/doc/man-pages/
다운로드: https://www.kernel.org/pub/linux/docs/man-pages/man-pages-5.05.tar.xz
MD5 sum: da25a4f8dfed0a34453c90153b98752d

• Meson (0.53.1) - 1,516 KB:
홈페이지: https://mesonbuild.com
다운로드: https://github.com/mesonbuild/meson/releases/download/0.53.1/meson-0.53.1.tar.gz
MD5 sum: 9bf73f7b5a2426a7c8674a809bb8cae2

• MPC (1.1.0) - 685 KB:
홈페이지: http://www.multiprecision.org/
다운로드: https://ftp.gnu.org/gnu/mpc/mpc-1.1.0.tar.gz
MD5 sum: 4125404e41e482ec68282a2e687f6c73

- MPFR (4.0.2) - 1,409 KB:
홈페이지: https://www.mpfr.org/
다운로드: http://www.mpfr.org/mpfr-4.0.2/mpfr-4.0.2.tar.xz
MD5 sum: 320fbc4463d4c8cb1e566929d8adc4f8
- Ninja (1.10.0) - 206 KB:
홈페이지: https://ninja-build.org/
다운로드: https://github.com/ninja-build/ninja/archive/v1.10.0/ninja-1.10.0.tar.gz
MD5 sum: cf1d964113a171da42a8940e7607e71a
- Ncurses (6.2) - 3,346 KB:
홈페이지: http://www.gnu.org/software/ncurses/
다운로드: http://ftp.gnu.org/gnu/ncurses/ncurses-6.2.tar.gz
MD5 sum: e812da327b1c2214ac1aed440ea3ae8d
- OpenSSL (1.1.1d) - 8,639 KB:
홈페이지: https://www.openssl.org/
다운로드: https://www.openssl.org/source/openssl-1.1.1d.tar.gz
MD5 sum: 3be209000dbc7e1b95bcdf47980a3baa
- Patch (2.7.6) - 766 KB:
홈페이지: https://savannah.gnu.org/projects/patch/
다운로드: http://ftp.gnu.org/gnu/patch/patch-2.7.6.tar.xz
MD5 sum: 78ad9937e4caadcba1526ef1853730d5
- Perl (5.30.1) - 12,078 KB:
홈페이지: https://www.perl.org/
다운로드: https://www.cpan.org/src/5.0/perl-5.30.1.tar.xz
MD5 sum: f399f3aaee90ddcff5eadd3bccdaacc0
- Pkg-config (0.29.2) - 1,970 KB:
홈페이지: https://www.freedesktop.org/wiki/Software/pkg-config
다운로드: https://pkg-config.freedesktop.org/releases/pkg-config-0.29.2.tar.gz
MD5 sum: f6e931e319531b736fadc017f470e68a
- Procps (3.3.15) - 884 KB:
홈페이지: https://sourceforge.net/projects/procps-ng
다운로드: https://sourceforge.net/projects/procps-ng/files/Production/procps-ng-3.3.15.tar.xz
MD5 sum: 2b0717a7cb474b3d6dfdeedfbad2eccc
- Psmisc (23.2) - 297 KB:
홈페이지: http://psmisc.sourceforge.net/
다운로드: https://sourceforge.net/projects/psmisc/files/psmisc/psmisc-23.2.tar.xz
MD5 sum: 0524258861f00be1a02d27d39d8e5e62
- Python (3.8.1) - 17,411 KB:
홈페이지: https://www.python.org/
다운로드: https://www.python.org/ftp/python/3.8.1/Python-3.8.1.tar.xz
MD5 sum: b3fb85fd479c0bf950c626ef80cacb57
- Python Documentation (3.8.1) - 6,374 KB:
다운로드: https://www.python.org/ftp/python/doc/3.8.1/python-3.8.1-docs-html.tar.bz2
MD5 sum: edc8c97f9680373fcc1dd952f0ea7fcc

- Readline (8.0) - 2,907 KB:
홈페이지: https://tiswww.case.edu/php/chet/readline/rltop.html
다운로드: http://ftp.gnu.org/gnu/readline/readline-8.0.tar.gz
MD5 sum: 7e6c1f16aee3244a69aba6e438295ca3
- Sed (4.8) - 1,317 KB:
홈페이지: http://www.gnu.org/software/sed/
다운로드: http://ftp.gnu.org/gnu/sed/sed-4.8.tar.xz
MD5 sum: 6d906edfdb3202304059233f51f9a71d
- Shadow (4.8.1) - 1,574 KB:
다운로드: https://github.com/shadow-maint/shadow/releases/download/4.8.1/shadow-4.8.1.tar.xz
MD5 sum: 4b05eff8a427cf50e615bda324b5bc45
- Sysklogd (1.5.1) - 88 KB:
홈페이지: http://www.infodrom.org/projects/sysklogd/
다운로드: http://www.infodrom.org/projects/sysklogd/download/sysklogd-1.5.1.tar.gz
MD5 sum: c70599ab0d037fde724f7210c2c8d7f8
- Sysvinit (2.96) - 120 KB:
홈페이지: https://savannah.nongnu.org/projects/sysvinit
다운로드: http://download.savannah.gnu.org/releases/sysvinit/sysvinit-2.96.tar.xz
MD5 sum: 48cebffebf2a96ab09bec14bf9976016
- Tar (1.32) - 2,055 KB:
홈페이지: http://www.gnu.org/software/tar/
다운로드: http://ftp.gnu.org/gnu/tar/tar-1.32.tar.xz
MD5 sum: 83e38700a80a26e30b2df054e69956e5
- Tcl (8.6.10) - 9,907 KB:
홈페이지: http://tcl.sourceforge.net/
다운로드: https://downloads.sourceforge.net/tcl/tcl8.6.10-src.tar.gz
MD5 sum: 97c55573f8520bcab74e21bfd8d0aadc
- Texinfo (6.7) - 4,237 KB:
홈페이지: http://www.gnu.org/software/texinfo/
다운로드: http://ftp.gnu.org/gnu/texinfo/texinfo-6.7.tar.xz
MD5 sum: d4c5d8cc84438c5993ec5163a59522a6
- Time Zone Data (2019c) - 383 KB:
홈페이지: https://www.iana.org/time-zones
다운로드: https://www.iana.org/time-zones/repository/releases/tzdata2019c.tar.gz
MD5 sum: f6987e6dfdb2eb83a1b5076a50b80894
- Udev-lfs Tarball (udev-lfs-20171102) - 11 KB:
다운로드: http://anduin.linuxfromscratch.org/LFS/udev-lfs-20171102.tar.xz
MD5 sum: 27cd82f9a61422e186b9d6759ddf1634
- Util-linux (2.35.1) - 5,018 KB:
홈페이지: http://freecode.com/projects/util-linux
다운로드: https://www.kernel.org/pub/linux/utils/util-linux/v2.35/util-linux-2.35.1.tar.xz
MD5 sum: 7f64882f631225f0295ca05080cee1bf

• Vim (8.2.0190) - 14,406 KB:
홈페이지: https://www.vim.org
다운로드: http://anduin.linuxfromscratch.org/LFS/vim-8.2.0190.tar.gz
MD5 sum: f5337b1170df90e644a636539a0313a3

참고
vim의 버전은 매일 바뀐다. 최신 버전을 구하려면 https://github.com/vim/vim/releases을 방문하라.

• XML::Parser (2.46) - 249 KB:
홈페이지: https://github.com/chorny/XML-Parser
다운로드: https://cpan.metacpan.org/authors/id/T/TO/TODDR/XML-Parser-2.46.tar.gz
MD5 sum: 80bb18a8e6240fcf7ec2f7b57601c170

• Xz Utils (5.2.4) - 1030 KB:
홈페이지: https://tukaani.org/xz
다운로드: https://tukaani.org/xz/xz-5.2.4.tar.xz
MD5 sum: 003e4d0b1b1899fc6e3000b24feddf7c

• Zlib (1.2.11) - 457 KB:
홈페이지: https://www.zlib.net/
다운로드: https://zlib.net/zlib-1.2.11.tar.xz
MD5 sum: 85adef240c5f370b308da8c938951a68

• Zstd (1.4.4) - 1,903 KB:
홈페이지: https://facebook.github.io/zstd/
다운로드: https://github.com/facebook/zstd/releases/download/v1.4.4/zstd-1.4.4.tar.gz
MD5 sum: 487f7ee1562dee7c1c8adf85e2a63df9

이 패키지들의 총 크기: 약 398 MB

3.3. 필요한 패치들

패키지 외에도 몇 가지 패치가 필요하다. 이 패치들은 관리자가 수정해야 하는 패키지의 오류들을 수정한다. 패치는 패키지를 사용하기 쉽게 하기 위해 약간의 수정도 한다. LFS 시스템을 구축하려면 다음과 같은 패치가 필요하다:

• Bash Upstream Fixes Patch - 22 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/bash-5.0-upstream_fixes-1.patch
MD5 sum: c1545da2ad7d78574b52c465ec077ed9

• Bzip2 Documentation Patch - 1.6 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/bzip2-1.0.8-install_docs-1.patch
MD5 sum: 6a5ac7e89b791aae556de0f745916f7f

• Coreutils Internationalization Fixes Patch - 168 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/coreutils-8.31-i18n-1.patch
MD5 sum: a9404fb575dfd5514f3c8f4120f9ca7d

• Glibc FHS Patch - 2.8 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/glibc-2.31-fhs-1.patch
MD5 sum: 9a5997c3452909b1769918c759eff8a2

- Kbd Backspace/Delete Fix Patch - 12 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/kbd-2.2.0-backspace-1.patch
MD5 sum: f75cca16a38da6caa7d52151f7136895
- Sysvinit Consolidated Patch - 2.4 KB:
다운로드: http://www.linuxfromscratch.org/patches/lfs/9.1/sysvinit-2.96-consolidated-1.patch
MD5 sum: 4900322141d493e74020c9cf437b2cdc

이 패치들의 총 크기: 약 208.8 KB

위의 필수 패치 외에도, LFS 커뮤니티에 의해 제작된 많은 선택적 패치가 있다. 이러한 선택적 패치는 사소한 문제를 해결하거나 기본적으로 활성화되지 않은 기능을 활성화한다. http://www.linuxfromscratch.org/patches/downloads/에 있는 패치 데이터베이스를 숙독하고 시스템 요구 사항에 맞는 추가 패치를 적용하라.

4장. 마지막 준비

4.1. 도입

이 장에서는 임시 시스템 구축을 준비하기 위해 몇 가지 추가 작업을 수행할 것이다. 임시 도구 설치를 위해 $LFS에 디렉토리를 생성하고, 위험을 줄이기 위해 권한이 약한 사용자를 추가하며, 해당 사용자에게 적합한 빌드 환경을 만들 것이다. 또한 LFS 패키지를 구축하는 데 걸리는 시간 단위, 다시말해 “SBU”를 설명하고 패키지 테스트 스위트에 대한 정보를 설명할 것이다.

4.2. $LFS/tools 디렉토리 생성

5장에서 컴파일될 모든 프로그램들은 6장에서 컴파일될 프로그램들과 따로 분류하기 위해 $LFS/tools 아래에 설치할 것이다. 여기서 컴파일된 프로그램들은 임시 도구이고 최종 LFS 시스템에 속하지 않는다. 이 프로그램들을 별도의 디렉토리에 보관함으로써 나중에 사용 후 쉽게 폐기할 수 있다. 또 이 프로그램들이 호스트 작업 디렉토리에 머무르는 일을 방지한다(5장에서 쉽게 저지르는 실수다).

다음을 root로 실행하여 필요한 디렉토리를 생성하라:

```
mkdir -v $LFS/tools
```

다음 단계는 호스트 시스템에 /tools symlink를 만드는 것이다. 이것은 LFS 파티션에 새로 생성된 디렉토리를 가리킬 것이다. root로 이 명령도 실행 하라:

```
ln -sv $LFS/tools /
```

참고

위의 명령은 오타가 난 것이 아니다. ln 명령은 여러가지 문법을 가지므로 오류라고 생각할 수 있는 내용을 보고하기 전에 info coreutils ln과 ln(1)을 반드시 확인하라.

생성된 symlink는 컴파일된 툴체인이 항상 /tools를 참조할 수 있게 하는데, 이것은 컴파일러, 어셈블러, 그리고 링커가 5장(아직까지 호스트의 일부 도구를 사용하는 시점)과 그 이후(LFS 파티션으로 “chroot”한 시점)에도 작동한다는 것을 의미한다.

4.3. LFS 유저 추가

root 유저로 로그인한 상태 에서는, 한 번의 실수로 시스템이 손상되거나 파괴될 수 있다. 따라서 다음 장에서는 일반 유저로 패키지를 빌드할 것을 권장한다. 당신만의 유저 이름을 사용할 수도 있지만, 깨끗한 작업 환경을 쉽게 설정할 수 있도록 lfs라는 새 사용자를 같은 이름의 lfs로 명명된 새 그룹의 구성원으로 생성하고 설치 절차 중에는 이 유저를 사용하라. 새 사용자를 추가하려면 root로 다음 명령을 실행하라:

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
```

명령 줄 옵션들의 의미:

-s /bin/bash
: lfs 유저의 기본 셸을 bash로 지정한다.

-g lfs
lfs 유저를 lfs 그룹에 추가한다.

-m
lfs 유저를 위한 홈 디렉토리를 생성한다.

-k /dev/null
이 매개변수는 홈 디렉토리 기본 폴더들을 가져오는 위치를 null 장치로 특정하여 스켈레톤 디렉토리(기본값은 /etc/skel)로부터의 폴더 복사를 방지한다.

lfs
생성된 그룹과 유저의 실제 이름이다.

lfs로 로그인하려면 (root에서 lfs 유저로 전환할 때와 같이 lfs 유저의 비밀번호가 필요 없는 경우와는 대조적으로), 다음과 같이 lfs의 암호를 지정하라:

```
passwd lfs
```

$LFS/tools에 대한 전체 접근 권한을 얻기 위해 lfs에게 디렉토리 소유권을 부여한다:

```
chown -v lfs $LFS/tools
```

권장한 방법대로 별도의 작업 디렉토리를 생성했다면 lfs 유저에게 이 디렉토리의 소유권을 부여하라:

```
chown -v lfs $LFS/sources
```

참고

일부 호스트 시스템에서는 다음 명령이 제대로 완료되지 않고 lfs 유저로의 로그인을 백그라운드로 일시 중지시킨다. "lfs:~$" 프롬프트가 즉시 나타나지 않으면 fg 명령을 입력하면 문제가 해결된다.

다음은 lfs 유저로의 로그인이다. 이 작업은 가상 콘솔이나 디스플레이 관리자를 통해 수행하거나 다음과 같은 대체 사용자 명령을 사용해 수행할 수 있다:

```
su - lfs
```

“-”는 su에게 비-로그인 셸과 달리 로그인 셸로 실행하도록 지시한다. 이 두 종류의 셸 사이의 차이는 bash(1)과 info bash에서 자세히 찾을 수 있다.

4.4. 환경 구축

bash 셸에 대한 두 개의 새 시작 파일을 만들어서 작업하기 좋은 환경을 설정한다. lfs 유저로 로그인하고 다음 명령을 실행하여 새 .bash_profile을 생성하라:

```
cat > ~/.bash_profile << "EOF"
exec env -i HOME=$HOME TERM=$TERM PS1='₩u:₩w₩$ ' /bin/bash
EOF
```

lfs 유저로 로그인하면, 기본 셸은 보통 호스트의 (여러 설정과 환경 변수가 들어있는)/etc/profile과 .bash_profile을 차례로 읽는 로그인 셸이다. .bash_profile파일의 exec env -i.../bin/bash 명령은 HOME, TERM, 그리고 PS1 변수를 제외하고 실행 중인 셸을 완전히 비어 있는 환경으로 대체한다. 이는 환경 변수들이 호스트 시스템에서 빌드 환경으로 의도치 않게 유출될 일이 없게 한다. 이런 테크닉으로 깨끗한 환경을 유지한다는 목표를 달성할 수 있다.

셸의 새 인스턴스는 /etc/profile이나 .bash_profile을 읽지 않고 .bashrc를 읽는 비-로그인 셸이다. 이제 .bashrc 파일을 만든다:

```
cat > ~/.bashrc << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LFS_TGT=$(uname -m)-lfs-linux-gnu
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL LFS_TGT PATH
EOF
```

set +h 명령은 bash의 해시함수를 끈다. 해싱은 일반적으로는 유용한 기능이다—bash는 PATH를 검색하고 다시 같은 실행 파일을 검색하는 것을 방지하고자 실행 파일들의 전체 경로를 기억하는 데에 해시 테이블을 사용한다. 그러나, 이 새로운 도구들은 설치하자마자 바로 사용해야 한다. 해시함수를 끄면 셸은 프로그램이 실행될 때 항상 PATH를 검색할 것이다. 이러면 셸은 다른 위치에 있는 동일한 프로그램의 기존 버전을 불러오지 않고 $LFS/tools에서 새로 컴파일되어 사용 가능해진 도구를 찾을 것이다.

사용자 파일 생성 마스크(umask)를 022로 설정하면 새로 생성된 파일과 디렉토리가 오로지 소유자만 쓰기 가능하면서 누구나 읽고 실행 가능한 상태가 된다. (기본 모드가 open(2) 시스템 호출에 의해 쓰여진다고 가정하면, 새 파일은 권한 모드 644를, 디렉토리는 755를 가진다.)

LFS 변수는 선택한 마운트 지점으로 설정되어야 한다.

LC_ALL 변수는 프로그램들의 언어 설정을 담당하여 메시지들을 지정된 국가의 언어로 내보낸다. LC_ALL을 “POSIX”나 “C”(둘의 결과는 같다)로 설정하면 chroot 환경에서 문제가 생기지 않을 것이다.

LFS_TGT 변수는 크로스 컴파일러와 링커를 빌드할 때, 그리고 임시 툴체인을 크로스 컴파일할 때 기본 지원은 아니지만 호환되는 시스템을 설명하는 데 사용된다. 자세한 설명은 5.2절. “툴체인 기술 노트”을 참조하라.

/tools/bin을 원래 PATH보다 앞에 두면, 5장에서 설치된 모든 프로그램들이 설치되자마자 셸에 인식된다. 이것은 해싱을 끄는 것과 더불어 5장에서 같은 프로그램이 공존할 때 호스트로부터 기존 프로그램들을 사용하는 실수를 방지한다.

끝으로, 방금 만든 사용자 프로필을 적용하여 임시 도구 빌드 환경의 준비를 마친다:

```
source ~/.bash_profile
```

4.5. SBU에 대해

많은 사람들이 각 패키지를 컴파일하고 설치하는 데 대략 얼마나 걸리는지 미리 알고 싶어한다. Linux From Scratch는 아주 다양한 시스템에 구축될 수 있기 때문에 정확한 시간 추정치를 제공하는 것은 불가능하다. 가장 큰 패키지(Glibc)는 가장 빠른 시스템에서는 약 20분이 걸리지만 느린 시스템에서는 최대 3일이 걸릴 수 있다! 실제 시간을 이야기하는 대신 표준 빌드 단위(SBU)측정을 사용할 것이다.

SBU는 다음과 같다. 이 책에서 처음으로 컴파일하는 패키지는 5장에서의 Binutils이다. 이 패키지를 컴파일하는 데 걸리는 시간이 표준 빌드 단위 또는 SBU로 언급될 단위이다. 다른 컴파일 시간은 모두 이 시간에 비례하여 표현될 것이다.

예를 들어 컴파일 시간이 4.5SBU인 패키지가 있다고 하자. 이는 시스템이 Binutils를 컴파일하고 설치하는 데 10분이 걸렸을 경우, 이 예시 패키지를 빌드하는데 대략 45분이 소요된다는 것을 의미한다. 다행히도 대부분의 빌드 시간은 Binutils보다 적게 걸린다.

일반적으로 SBU는 호스트 시스템의 GCC 버전을 포함한 많은 요인에 따라 다르기 때문에 완전히 정확한 것은 아니다. 패키지를 설치하는 데 얼마나 걸릴지 알리기 위해 표기되어 있지만, 경우에 따라 그 오차는 수십 분까지 차이날 수 있다.

참고

프로세서(또는 코어)가 여러 개인 최신 시스템의 경우, 환경 변수를 설정하거나 make 프로그램에 가용한 프로세서 수를 입력해 "병렬 make"를 진행함으로써 패키지의 컴파일 시간을 줄일 수 있다. 예를 들어 Core2Duo는 다음 명령으로 2코어를 동시에 사용할 수 있다:

```
export MAKEFLAGS='-j 2'
```

또는 그냥 바로 빌드하려면:

```
make -j2
```

여러 개의 프로세서를 이런 식으로 사용할 경우, 책에 있는 SBU 단위는 기존보다 훨씬 더 차이날 것이다. 경우에 따라 make 단계에서 에러가 날 수도 있다. 서로 다른 처리 작업이 교차하기 때문에 빌드 과정의 출력을 분석하기도 더 어려울 것이다. 빌드 단계에서 문제가 발생하면 단일 프로세서 빌드로 되돌아가서 오류 메시지를 정확히 분석하라.

4.6. 테스트 스위트에 대해

대부분의 패키지들은 테스트 스위트를 제공한다. 모든 것이 정확하게 컴파일됐는지 알 수 있는 “안전성 검사”를 하려면 새로 빌드된 패키지의 테스트 스위트를 실행하는 것이 좋다. 테스트 스위트가 일련의 검사들을 통과하면 보통은 패키지가 개발자가 의도한 대로 작동하고 있다는 의미이다. 그러나 그렇다고 해서 버그가 아예 없다는 보장은 없다.

어떤 테스트 스위트는 다른 것보다 더 중요하다. 예를 들어 핵심 툴체인 패키지들—GCC, Binutils와 Glibc—을 위한 테스트 스위트들은 시스템에서 중요한 역할을 하기 때문에 가장 중요하다. GCC와 Glibc의 테스트 스위트는 특히 느린 하드웨어에서 완료하는 데 매우 오랜 시간이 걸릴 수 있지만 강력하게 권장한다.

참고

경험에 따르면 5장에서 테스트 스위트를 돌리는 것은 별로 얻을만한 것이 없었다. 호스트 시스템이 해당 장의 테스트에 항상 영향을 미쳐 계속해서 설명할 수 없는 오류가 발생한다는 사실을 부정할 수 없다. 5장에서 빌드한 도구들은 일시적이고 결국 폐기할 것이기 때문에 일반 독자들에게는 5장의 테스트 스위트를 행하는 것을 권장하지 않는다. 이 테스트 스위트들을 실행하는 절차들은 테스터와 개발자들을 위해 제시하지만 엄밀히 말해 선택 사항일 뿐이다.

Binutils 및 GCC에 대한 테스트 스위트 실행에 관한 흔한 문제는 가상 터미널(PYTs)의 부족이다. 이로 인해 많은 테스트 실패를 겪을 수 있다. 이런 현상은 여러 이유로 발생할 수 있지만, 가장 가능성이 높은 원인은 호스트 시스템에 devpts 파일 시스템이 올바르게 설정되어 있지 않기 때문이다. 이 문제는 http://www.linuxfromscratch.org/lfs/faq.html#no-ptys에서 자세히 설명한다.

이따금씩 패키지 테스트 스위트가 실패하겠지만 그것은 개발자들이 그다지 치명적이지 않다고 여기기 때문이다. http://www.linuxfromscratch.org/lfs/build-logs/9.1/에서 로그들을 참고해서 그런 테스트 실패가 예상된 케이스인지 확인하라. 이 사이트는 이 책의 모든 테스트에 대해 유효하다.

5장. 임시 시스템 구축

5.1. 도입

이 장에서는 최소한의 리눅스 시스템을 구축하는 방법을 설명한다. 이 시스템은 6장에서 최종 LFS 시스템을 구축하기에 충분한 도구들을 포함해서 최소 환경보다 사용자 편의성이 좋은 작업 환경을 만들어 줄 것이다.

이 최소한의 시스템을 구축하는 데는 두 단계가 있다. 첫 번째 단계는 호스트에 독립적인 새로운 툴체인(컴파일러, 어셈블러, 링커, 라이브러리 및 몇 가지 유용한 유틸리티들)을 빌드하는 것이다. 두 번째 단계는 이 툴체인을 사용하여 다른 필수 도구들을 빌드한다.

이 장에서 컴파일될 파일들은 다음 장에서 설치될 파일과 호스트 작업 디렉토리로부터 분리시키기 위해 $LFS/tools 디렉토리에 설치될 것이다. 여기서 컴파일된 패키지들은 잠깐 쓰일 것이기 때문에 곧이어 만들어질 LFS 시스템을 오염시켜선 안 된다.

5.2. 툴체인 기술 노트

이 절에서는 전체 빌드 방법에 대한 몇 가지 해석과 기술적 세부 사항을 설명한다. 이 절의 모든 것을 당장 이해할 필요는 없다. 이 정보들의 대부분은 실제 빌드를 하고 나서야 더 명확해질 것이다. 이 절은 진행 중에 언제든지 참고하라.

5장의 전반적인 목표는 앞서 언급한 호스트 시스템으로부터 분리할 수 있는 도구들을 포함하는 임시 공간을 구축하는 것이다. chroot를 사용함으로써, 나머지 장에 있는 명령들이 이 환경에 포함되어 대상 LFS 시스템을 깨끗하고 문제없이 구축할 수 있다. 빌드 절차는 새로운 독자를 위해 위험을 최소화하고 동시에 가장 교육적인 가치들을 제공하도록 설계되었다.

참고

계속 진행하기 전에, 작업 중인 플랫폼의 명칭, 흔히 대상 트리플렛(target triplet)이라 칭하는 것을 알고 있어야 한다. 대상 트리플렛의 명칭을 구하는 간단한 방법은 많은 패키지의 소스와 함께 제공되는 config.guess 스크립트를 실행하는 것이다. Binutils 소스를 압축 해제하고 이 스크립트를 실행하라: ./config.guess 그리고 출력 결과를 기록하라. 예를 들어 32비트 인텔 프로세서의 경우 출력은 i686-pc-linux-gnu가 된다. 64비트 시스템에서는 x86_64-pc-linux-gnu일 것이다.

추가로 흔히 동적 로더(Binutils의 일부인 표준 링커 ld와 혼동하지 말라)라 불리는, 플랫폼의 동적 링커의 이름도 알고 있어야 한다. Glibc가 제공하는 동적 링커는 프로그램이 필요로 하는 공유 라이브러리를 찾아 불러오고, 실행할 프로그램을 준비한 후 실행한다. 32비트 인텔 시스템의 동적 링커 이름은 ld-linux.so.2(64비트 시스템의 경우 ld-linux-x86-64.so.2)일 것이다. 동적 링커의 이름을 알아내는 확실한 방법은 호스트 시스템에서 아무 바이너리나 검사하는 것이다: readelf -l <바이너리의 이름> | grep interpreter 을 실행하고 출력 결과를 기록하라. 모든 플랫폼을 다루는 신뢰할만한 레퍼런스는 Glibc 소스 트리의 최상위에 있는 shlib-versions 파일에 있다.

5장의 빌드 방법 작동 방식에 대한 몇 가지 핵심 기술적 요점:

- LFS_TGT 변수를 통해 대상 트리플렛의 "벤더" 항을 변경하여 작업 플랫폼의 이름을 약간 조정하면, Binutils 및 GCC의 첫 번째 빌드가 호환 가능한 크로스 링커와 크로스 컴파일러를 생성한다. 다른 아키텍쳐에 대한 바이너리들을 만드는 대신 크로스 링커와 크로스 컴파일러로 현재 하드웨어와 호환되는 바이너리를 생성할 것이다.
- 임시 라이브러리들은 크로스 컴파일된다. 크로스 컴파일러는 본질적으로 호스트 시스템의 어떤 것에도 의존할 수 없기 때문에, 이 방법은 호스트의 헤더나 라이브러리가 새 도구에 합쳐질 가능성을 줄임으로써 대상 시스템과 뒤섞일 위험을 제거한다. 또 크로스 컴파일을 하면 32비트 및 64비트 라이브러리를 64비트 하드웨어에 모두 빌드할 수 있다.

• GCC 소스를 조심히 조작해서 컴파일러에게 어떤 대상 동적 링커를 사용할지 정할 수 있다.

GCC와 Glibc의 configure를 실행하면 어셈블러와 링커에서 다양한 기능 테스트를 거쳐 어떤 소프트웨어 기능을 활성화할지 결정하기 때문에 (어셈블러와 링커들이 포함된) Binutils를 먼저 설치한다. 이것은 보기보다 중요하다. 잘못 설정된 GCC나 Glibc는 툴체인의 미묘한 고장으로 이어질 수 있으며, 이러면 그 영향이 전체 시스템 빌드가 끝날 때까지 나타나지 않을 수 있다. 보통 테스트 스위트 실패는 이미 너무 많은 작업을 수행한 채로 뒤늦게 알아채기 전에 이 오류를 알려준다.

Binutils는 어셈블러와 링커를 /tools/bin과 /tools/$LFS_TGT/bin 두 위치에 설치한다. 한 위치의 도구는 다른 위치에 있는 도구로의 하드링크이다. 링커는 라이브러리 검색 순서가 중요하다. 자세한 정보는 ld에 --verbose 플래그를 붙여 실행하면 얻을 수 있다. 예를 들어 ld --verbose | grep SEARCH는 현재 검색 경로와 그 순서를 보여준다. 더미 프로그램을 컴파일하고 링커에 --verbose 옵션을 전달하면 어떤 파일들이 ld에 의해 링크되었는지 알 수 있다. 예를 들어 gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded은 링크 중에 성공적으로 열린 모든 파일들을 보여줄 것이다.

다음으로 설치될 패키지는 GCC이다. configure 실행 중에 볼 수 있는 예시이다:

```
checking what assembler to use... /tools/i686-lfs-linux-gnu/bin/as
checking what linker to use... /tools/i686-lfs-linux-gnu/bin/ld
```

이것은 위에 언급한 이유때문에 중요하다. 이는 또한 GCC의 configure 스크립트가 어떤 도구를 사용할지 찾느라 PATH 디렉토리를 검색하지 않는다는 것을 알 수 있다. 단, gcc 자체의 실제 작동 중에는 반드시 동일한 검색 경로가 사용되는 것은 아니다. gcc가 어떤 표준 링커를 사용할지 알아보려면 gcc -print-prog-name=ld를 실행하라.

더미 프로그램을 컴파일하면서 gcc에 -v 명령줄 옵션을 전달하면 자세한 정보를 얻을 수 있다. 예를 들어 gcc -v dummy.c는 gcc에 포함된 검색 경로 및 순서, 전처리기, 검파일, 어셈블리 단계에 대한 자세한 정보를 표시한다.

다음으로 설치될 것은 순수한 리눅스 API 헤더이다. 이를 통해 표준 C 라이브러리(Glibc)가 리눅스 커널에서 제공할 기능들과 상호 작용할 수 있다.

다음으로 설치될 패키지는 Glibc이다. Glibc 빌드에 있어 가장 중요한 것은 컴파일러, 바이너리 도구, 커널 헤더들이다. Glibc는 항상 configure 스크립트에 전달되는 --host 매개 변수에 관련된 컴파일러를 사용하기 때문에, 컴파일러는 대게 문제가 되지 않는다; 이 책의 컴파일러는 i686-lfs-linux-gnu-gcc이다. 바이너리 도구와 커널 헤더들의 경우는 조금 더 복잡하다. 그러니 위험을 무릅쓰지 말고 올바른 선택을 하려면 사용 가능한 configure 설정을 사용하라. configure 실행을 마친 뒤, glibc-build 디렉토리에서 config.make 파일을 열어 모든 중요한 세부사항을 확인하라. CC="i686-lfs-gnu-gcc"로 어떤 바이너리 도구들을 사용할지 지정하고 -nostdinc와 -isystem 플래그로 컴파일러의 include 탐색 경로를 정한다. 이 항목들로부터 Glibc 패키지의 중요한 점을 알 수 있다. Glibc 패키지는 빌드 환경에 있어서 매우 독립적이고 일반적으로 툴체인 기본값에 의존하지 않는다.

새로 빌드된 Binutils 2단계에서는 --with-lib-path configure 옵션을 활용하여 ld의 라이브러리 탐색 경로를 지정할 수 있다.

GCC의 재구축을 위해서는 새로운 동적 링커를 사용하도록 소스도 수정해야 한다. 그렇게 하지 않으면 GCC 프로그램 안에 호스트 시스템의 /lib 디렉토리에 있는 동적 링커의 이름이 포함되어 호스트에서 독립하려는 목표를 달성할 수 없다. 이 때부터 핵심 툴체인은 독립적으로 작동할 수 있게 된다. 나머지 5장 패키지는 모두 /tools의 새로운 Glibc를 기반으로 구축된다.

6장에서 chroot 환경에 들어가면, 위에서 언급한대로 독립적으로 작동할 수 있는 덕에 Glibc 패키지가 가장 먼저 설치된다. 이 Glibc가 /usr에 설치되고 나면 툴체인 기본값을 신속히 바꾸고 나머지 목표 LFS 시스템 구축을 진행할 것이다.

5.3. 일반적인 컴파일 절차

패키지를 빌드할 때는 지침들이 내포하는 몇 가지 가정들이 있다:

- 몇몇 패키지는 컴파일 전에 패치를 적용하겠지만, 문제를 일으키지 않기 위해 필요할 때만 패치를 적용한다. 패치들은 보통 이 장과 다음 장 모두에서 필요하지만, 때로는 한 부분에서만 필요할 때도 있다. 따라서 다운로드한 패치에 대한 지침이 누락된 것 같아도 걱정하지 마라. 패치를 적용할 때 offset이나 fuzz에 관한 경고 메시지가 나타날 수도 있다. 그래도 패치가 성공적으로 적용될테니 이러한 경고에 대해서는 걱정할 필요 없다.
- 대부분의 패키지를 컴파일하는 동안, 화면을 뒤덮는 여러 경고가 있을 것이다. 이것들은 정상적인 것이며 무시해도 좋다. 이런 경고들은 말하자면—더 이상 사용되지 않지만 잘못되지는 않은 C/C++ 문법에 대한 경고이다. C 표준은 상당히 자주 바뀌는데 일부 패키지는 여전히 예전 표준을 사용하기 때문에 일어나는 일이다. 경고는 출력되겠지만, 문제가 생기진 않는다.
- 마지막으로 LFS 환경 변수가 올바르게 설정되었는지 재차 확인하라:

```
echo $LFS
```

  LFS 파티션의 마운트 지점 경로(예시를 따랐다면 /mnt/lfs)가 확실히 출력되는지 확인하라.
- 마지막으로, 두 가지 중요한 항목을 꼭 기억하라:

중요

이 빌드 절차는 심볼릭 링크를 포함한 호스트 시스템 요구사항이 다음과 같이 올바르게 설정되었다고 간주한다:

- bash를 셸로 사용한다.
- sh는 bash로 심볼릭 링크되었다.
- /usr/bin/awk는 gawk로 심볼릭 링크되었다.
- /usr/bin/yacc는 bison이나 bison을 실행하는 스몰 스크립트로 심볼릭 링크되었다.

중요

빌드 절차를 다시 강조한다:

1. 모든 소스들과 패치들을 /mnt/lfs/sources/같은 chroot 환경에서 접근할 수 있는 디렉토리에 두어라. 소스를 /mnt/lfs/tools/에 넣지 않도록 하라.
2. 소스들이 있는 디렉토리로 이동한다.
3. 각 패키지에 대해:
   a. tar 프로그램을 사용해서 빌드될 패키지를 추출한다. 5장에서는, 패키지를 추출할 때 lfs 유저로 로그인하라.
   b. 패키지를 추출해서 생성된 디렉토리로 이동한다.
   c. 패키지를 빌드하기 위해 책의 절차를 따른다.
   d. 소스들이 있는 디렉토리로 돌아간다.
   e. 달리 지시되지 않는 한 추출하고 난 원본 디렉토리를 삭제한다.

5.4. Binutils-2.34 - 1단계

Binutils 패키지는 링커, 어셈블러 및 객체 파일을 처리하기 위한 기타 도구를 포함한다.

예상 빌드 시간: 1 SBU
필요 디스크 공간: 625 MB

5.4.1. Cross Binutils 설치

참고

이전 절로 돌아가 주의 사항들을 되짚어보라. 중요하다고 표시된 항목들을 이해하면 앞으로의 많은 문제들을 줄일 수 있다.

Binutils를 가장 먼저 컴파일하는 것이 중요하다. Glibc와 GCC는 각자 어떤 기능을 활성화할지 정하기 위해 가용한 링커와 어셈블러를 통해 다양한 테스트를 하기 때문이다.

Binutils 문서에서는 Binutils를 전용 빌드 디렉토리에서 빌드할 것을 권장한다:

```
mkdir -v build
cd       build
```

참고

이 책에서 앞으로 나올 SBU 값을 의미있게 참고하려면, 설정에서 첫 번째 설치까지 이 패키지를 빌드하는 데 걸린 시간을 측정하라. 이 측정을 쉽게 하려면 명령어를 다음과 같이 time 명령어로 감싸라: time { ./configure ... && ... && make install; }.

참고

5장의 대략적인 빌드 SBU 값과 필요 디스크 공간은 테스트 스위트 데이터는 제외한 값이다.

이제 Binutils 컴파일을 준비한다:

```
../configure --prefix=/tools            ₩
             --with-sysroot=$LFS        ₩
             --with-lib-path=/tools/lib ₩
             --target=$LFS_TGT          ₩
             --disable-nls              ₩
             --disable-werror
```

Configure 옵션들의 의미:

--prefix=/tools
Configure 스크립트가 Binutils 프로그램들을 /tools 디렉토리에 설치하도록 한다.

--with-sysroot=$LFS
크로스 컴파일을 위해 빌드 시스템이 필요에 따라 $LFS에서 대상 시스템 라이브러리를 찾도록 지시한다.

--with-lib-path=/tools/lib
링커가 사용할 라이브러리 경로를 지정한다.

--target=$LFS_TGT
LFS_TGT 변수의 시스템 설명이 config.guess 스크립트에서 반환되는 값과 약간 다르기 때문에, configure 스크립트를 통해 크로스 링커 빌드를 위한 Binutils의 빌드 시스템을 조정한다.

--disable-nls
임시 도구에는 i18n이 필요하지 않으므로 다국어화를 비활성화한다.

--disable-werror
호스트 컴파일러로부터 나온 경고로 빌드가 멈추는 것을 방지한다.

계속해서 패키지 컴파일을 진행한다:

```
make
```

이제 컴파일이 끝났다. 원래대로라면 이제 테스트 스위트를 실행했겠지만, 아직은 테스트 스위트 프레임워크(Tcl, Expect 및 DejaGNU)가 아직 준비되지 않았다. 이 첫 번째 단계의 프로그램은 곧 두 번째 단계의 프로그램으로 대체될 것이기 때문에 지금 테스트를 실행해봤자 큰 이점은 없다.

x86_64에서 빌드 중이라면, 심볼릭 링크를 만들어서 툴체인의 안정성을 보장하라:

```
case $(uname -m) in
  x86_64) mkdir -v /tools/lib && ln -sv lib /tools/lib64 ;;
esac
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.18.2절. “Binutils의 내용.”에서 찾을 수 있다.

5.5. GCC-9.2.0 - 1단계

GCC 패키지는 C, C++ 컴파일러가 들어있는 GNU 컴파일러 컬렉션을 포함하고 있다.

예상 빌드 시간: 10 SBU
필요 디스크 공간: 3.1 GB

5.5.1. Cross GCC 설치

GCC는 이제 GMP, MPFR 그리고 MPC 패키지를 필요로 한다. 이러한 패키지들은 호스트 배포판에 포함되어 있지 않을 수 있으므로 GCC로 빌드될 것이다. GCC 빌드 과정에 쓰일 수 있도록 각 패키지를 GCC 소스 디렉토리에 압축을 풀고 생성된 디렉토리의 이름을 변경하라:

참고

이 장에 대해서 오해가 빈번하다. 이 절차들은 앞서 설명했듯이 다른 모든 장과 동일하다(패키지 빌드 절차). 먼저 sources 디렉토리에서 GCC tarball을 추출한 다음 생성된 디렉토리로 이동하라. 그래야만 아래 지침을 따를 수 있다.

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.2.0.tar.xz
mv -v gmp-6.2.0 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

다음 명령은 GCC의 기본 동적 링커 위치를 /tools에 설치된 링커로 변경한다. GCC의 포함 검색 경로에서도 /usr/include를 제거한다:

```
for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib₩(64₩)₩?₩(32₩)₩?/ld@/tools&@g' ₩
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

위의 내용이 어려워 보이면 조금 더 살펴보자. 먼저 gcc/config/linux.h, gcc/config/i386/linux.h 그리고 gcc/config/i368/linux64.h 파일들을, 파일명은 같고 접미사(확장자)가 “.orig”인 파일로 복사한다. 첫 번째 sed 표현식은 “/lib/ld”, “/lib64/ld” 또는 “/lib32/ld”의 모든 인스턴스 앞에 “/tools”를 붙이며, 두 번째 표현식은 “/usr”의 하드 코딩된 인스턴스를 대체한다. 그리고나서, 기본 시작파일 접두사를 파일 끝으로 변경하는 define 문을 추가한다. “/tools/lib/”의 끝에 “/”가 필요하다는 점에 유의하라. 마지막으로, touch를 사용하여, 복사된 파일의 타임 스탬프를 업데이트한다. cp -u와 함께 사용하면 명령이 실수로 두 번 실행되더라도 원본 파일이 예기치 않게 변경되지 않는다.

마지막으로, x86_64 호스트에서 64비트 라이브러리의 기본 디렉토리 이름을 “lib”로 설정한다:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' ₩
        -i.orig gcc/config/i386/t-linux64
 ;;
esac
```

GCC 문서는 전용 빌드 디렉토리에 GCC를 구축할 것을 권장한다:

```
mkdir -v build
cd       build
```

GCC 컴파일을 준비한다:

```
../configure                                       ₩
    --target=$LFS_TGT                              ₩
    --prefix=/tools                                ₩
    --with-glibc-version=2.11                      ₩
    --with-sysroot=$LFS                            ₩
    --with-newlib                                  ₩
    --without-headers                              ₩
    --with-local-prefix=/tools                     ₩
    --with-native-system-header-dir=/tools/include ₩
    --disable-nls                                  ₩
    --disable-shared                               ₩
    --disable-multilib                             ₩
    --disable-decimal-float                        ₩
    --disable-threads                              ₩
    --disable-libatomic                            ₩
    --disable-libgomp                              ₩
    --disable-libquadmath                          ₩
    --disable-libssp                               ₩
    --disable-libvtv                               ₩
    --disable-libstdcxx                            ₩
    --enable-languages=c,c++
```

Configure 옵션들의 의미:

--with-glibc-version=2.11
패키지가 호스트의 Glibc 버전과 호환되도록 지정한다. 호스트 시스템 요구사항에 명시된 최소 Glibc 버전으로 지정한다.

--with-newlib
아직 C 라이브러리를 사용할 수 없으므로 libgcc를 빌드할 때 inhibit_libc 상수가 정의되도록 한다. 이 상수는 libc가 필요한 어떤 코드도 컴파일되지 않게 한다.

--without-headers
완전한 크로스 컴파일러를 만들 때 GCC는 대상 시스템과 호환되는 표준 헤더가 필요하다. 우리의 목표를 위해선 이 헤더가 필요하지 않다. 이 옵션은 GCC가 그 헤더들을 찾지 않도록 한다.

--with-local-prefix=/tools
'local prefix'란 GCC가 로컬로 설치된 include 파일들을 검색하는 위치이다. 기본값은 /usr/local이다. 이 위치를 /tools로 설정하면 GCC의 검색 경로에 /usr/local이 포함되지 않는다.

--with-native-system-header-dir=/tools/include
기본적으로 GCC는 /usr/include에서 시스템 헤더를 검색한다. 보통 sysroot 변경과 함께 검색 위치가 $LFS/usr/include로 이동한다. 그러나 다음 두 절에서 설치될 헤더들은 $LFS/tools/include에 설치된다. 이 옵션은 GCC가 그들을 정확하게 찾을 수 있도록 한다. GCC의 두 번째 단계에서도 호스트 시스템의 헤더를 찾지 않도록 하기 위해 이 옵션이 그대로 쓰인다.

--disable-shared
GCC가 내부 라이브러리를 정적으로 링크하도록 강제한다. 호스트 시스템에 발생할 수 있는 문제를 방지하고자 함이다.

--disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx
이 옵션들은 각각 십진 부동소수점 확장, 스레딩, libatomic, libquadmath, libssp, libvtv 및 C++ 표준 라이브러리를 비활성화한다. 이러한 기능들은 크로스 컴파일러를 빌드할 때 컴파일에 실패하며 임시 libc를 크로스 컴파일하는 데에는 필요하지 않다.

--disable-multilib
x86_64에서 LFS는 아직 multilib 구성을 지원하지 않는다. 이 옵션은 x86에선 아무 영향이 없다.

--enable-languages=c,c++
C와 C++ 컴파일러만 빌드되도록 한다. 당장 필요한 언어는 이 둘 뿐이다.

다음으로 GCC를 컴파일하라:

```
make
```

이제 컴파일이 끝났다. 이 시점에서 보통 테스트 스위트를 실행하지만, 앞서 언급했듯 테스트 스위트 프레임워크는 아직 구현되어 있지 않다. 이 첫 단계의 프로그램은 곧 교체될 것이기 때문에 지금 테스트를 실행해봤자 얻을 수 있는 이점은 거의 없다.

패키지를 설치한다:

```
make install
```

이 패키지에 관한 자세한 정보는 6.25.2절. “GCC의 내용.”에서 찾을 수 있다.

5.6. Linux-5.5.3 API 헤더

리눅스-5.5.3.tar.xz의 리눅스 API 헤더는 Glibc에 쓰이는 커널 API를 담고있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 1 GB

5.6.1. 리눅스 API 헤더 설치

리눅스 커널은 시스템의 C 라이브러리(LFS에선 Glibc)가 사용할 애플리케이션 프로그래밍 인터페이스(API)를 공개해야한다. 이는 리눅스 커널 소스 tarball에 들어있는 여러 C 헤더 파일들을 검사하는 방식으로 이뤄진다.

패키지에 포함된 오래된 파일이 없도록 만든다:

```
make mrproper
```

이제 소스에서 사용자가 볼 수 있는 커널 헤더를 추출한다. 아직 사용할 수 없는 rsync가 필요하기 때문에, 권장된 make 대상 “headers_install”은 사용할 수 없다. 헤더들을 일단 ./usr에 둔 다음, 필요한 위치에 복사한다.

```
make headers
cp -rv usr/include/* /tools/include
```

이 패키지에 대한 자세한 정보는 6.7.2절. “리눅스 API 헤더의 내용.”에서 찾을 수 있다.

5.7. Glibc-2.31

Glibc 패키지는 메인 C 라이브러리를 포함하고 있다. 이 라이브러리는 메모리 할당, 디렉토리 검색, 파일 열기 및 닫기, 파일 읽기 및 쓰기, 문자열 처리, 패턴 대조, 산술 등의 기본 루틴을 제공한다.

예상 빌드 시간: 4.5 SBU
필요 디스크 공간: 896 MB

5.7.1. Glibc 설치

Glibc 문서에서는 전용 빌드 디렉토리에 Glibc를 빌드할 것을 권장한다:

```
mkdir -v build
cd       build
```

다음으로 Glibc 컴파일을 준비한다:

```
../configure                           ₩
      --prefix=/tools                  ₩
      --host=$LFS_TGT                  ₩
      --build=$(../scripts/config.guess) ₩
      --enable-kernel=3.2              ₩
      --with-headers=/tools/include
```

Configure 옵션들의 의미:

--host=$LFS_TGT, --build=$(../scripts/config.guess)
/tools의 크로스 링커와 크로스 컴파일러를 사용해서 Glibc의 빌드 시스템이 스스로를 크로스 컴파일하게 한다.

--enable-kernel=3.2
Glibc가 3.2 이상 리눅스 커널을 지원하는 라이브러리를 컴파일하도록 한다. 이전 커널에 대한 라이브러리는 사용할 수 없다.

--with-headers=/tools/include
앞서 tools 디렉토리에 설치된 헤더들을 가지고 컴파일하도록 한다. 커널이 어떤 기능들을 갖고 있는지 정확히 알고 그에 따른 최적화를 할 수 있다.

이 단계에서 다음과 같은 경고가 나타날 수 있다:

```
configure: WARNING:
*** These auxiliary programs are missing or
*** incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.
```

누락됐거나 호환되지 않는다는 msgfmt 프로그램은 보통 문제가 되지 않는다. 이 msgfmt 프로그램은 호스트 배포판이 제공하는 Gettext 패키지의 일부다.

참고

"병렬 make"로 빌드할 때 실패하는 경우가 있다는 보고가 있었다. 이럴 경우 "-j1" 옵션을 사용해서 make 명령을 다시 실행하라.

패키지를 컴파일한다:

```
make
```

패키지를 설치한다:

```
make install
```

경고

여기서 잠깐 멈추고 새로운 툴체인의 기본 기능(컴파일 및 링크)이 기대한 대로 잘 작동하는지 확인하는 것이 우선이다. 다음 명령으로 온전성 검사를 수행하라:

```
echo 'int main(){}' > dummy.c
$LFS_TGT-gcc dummy.c
readelf -l a.out | grep ': /tools'
```

모든 것이 제대로 작동한다면 오류가 없어야 하며 마지막 명령의 출력은 다음과 같을 것이다:

```
[Requesting program interpreter: /tools/lib64/ld-linux-x86-64.so.2]
```

32비트 시스템에서는 인터프리터 이름이 /tools/lib/ld-linux.so.2임을 참고하라.

만약 출력이 위와 같이 표시되지 않거나 출력이 전혀 없다면, 무언가 잘못된 것이다. 이전 단계로 돌아가 어디서 문제가 생겼는지 파악하고 수정하라. 이 문제는 계속 진행하기 전에 반드시 해결해야 한다.

모두 잘 됐다면, 테스트 파일을 정리하라:

```
rm -v dummy.c a.out
```

참고

다음 절에서 Binutils를 빌드하면 툴체인이 올바르게 빌드되었는지 다시금 확인하는 기회가 될 것이다. Binutils 빌드에 실패하면, 이전 Binutils, GCC, 또는 Glibc 중 어딘가 문제가 생겼음을 보여주는 것이다.

이 패키지에 대한 자세한 정보는 6.9.3절. “Glibc의 내용.”에서 찾을 수 있다.

5.8. Libstdc++ from GCC-9.2.0

Libstdc++은 표준 C++ 라이브러리이다. 이것은 C++ 코드를 컴파일할 때 필요하지만(GCC의 일부는 C++로 쓰여졌다), gcc-1단계를 빌드할 때는 설치를 미룰 수 밖에 없었다. 그 때는 아직 /tools에 없었던 glibc가 필요했기 때문이다.

예상 빌드 시간: 0.5 SBU
필요 디스크 공간: 878 MB

5.8.1. 대상 Libstdc++ 설치

참고

Libstdc++는 GCC 소스의 일부이다. GCC tarball을 먼저 풀고 gcc-9.2.0 디렉토리로 이동해야 한다.

Libstdc++을 위한 별도의 디렉토리를 생성하고 이동하라:

```
mkdir -v build
cd       build
```

Libstdc++ 컴파일을 준비한다:

```
../libstdc++-v3/configure           ₩
   --host=$LFS_TGT                  ₩
   --prefix=/tools                  ₩
   --disable-multilib               ₩
   --disable-nls                    ₩
   --disable-libstdcxx-threads      ₩
   --disable-libstdcxx-pch          ₩
   --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/9.2.0
```

Configure 옵션들의 의미:

--host=...
/usr/bin에 있는 컴파일러 대신 방금 만든 크로스 컴파일러를 사용하도록 지시한다.

--disable-libstdcxx-threads
GCC 1단계가 스레드를 지원하지 않기 때문에 C++ 스레드 라이브러리도 지원되지 않는다.

--disable-libstdcxx-pch
이 단계에서 필요하지 않은 사전 컴파일된 include 파일의 설치를 방지한다.

--with-gxx-include-dir=/tools/$LFS_TGT/include/c++/9.2.0
C++ 컴파일러가 표준 include 파일을 검색하는 위치이다. 일반적으로는 이 정보가 최상위 디렉토리로부터 Libstdc++ configure 옵션으로 자동 전달된다. 때문에 우리는 이 주소를 명시적으로 입력해야 한다.

Libstdc++를 컴파일한다:

```
make
```

라이브러리를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.25.2절. “GCC의 내용.”에서 찾을 수 있다.

5.9. Binutils-2.34 - 2단계

Binutils 패키지는 링커, 어셈블러 및 객체 파일을 처리하기 위한 기타 도구를 포함한다.

예상 빌드 시간: 1.1 SBU
필요 디스크 공간: 651 MB

5.9.1. Binutils 설치

다시 별도의 빌드 디렉토리를 생성한다:

```
mkdir -v build
cd       build
```

Binutils 컴파일을 준비한다:

```
CC=$LFS_TGT-gcc              ₩
AR=$LFS_TGT-ar               ₩
RANLIB=$LFS_TGT-ranlib       ₩
../configure                 ₩
   --prefix=/tools           ₩
   --disable-nls             ₩
   --disable-werror          ₩
   --with-lib-path=/tools/lib ₩
   --with-sysroot
```

새 Configure 옵션들의 의미:

CC=$LFS_TGT-gcc AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib
이것은 실사용할 Binutils의 네이티브 빌드이기 때문에, 이 옵션을 설정하여 빌드 시스템으로 하여금 호스트 시스템이 아닌 크로스 컴파일러와 관련 도구들을 사용하게끔 한다.

--with-lib-path=/tools/lib
Binutils를 컴파일하는 동안 라이브러리 검색 경로를 지정하도록 configure 스크립트에 지시해서 /tools/lib가 링커에 전달되도록 한다. 이렇게 해서 링커가 호스트의 라이브러리 디렉토리를 검색하는 것을 방지할 수 있다.

--with-sysroot
기본(존재하지 않는) sysroot 디렉토리 /tools/$LFS_TGT/sys-root를 정의한다. 링커의 명령줄에 명시된 다른 공유 객체에 필요한 공유 객체를 찾을 때 유용하다. 이러한 객체들은 <sysroot>/etc/ld.so.conf에 나열된 디렉토리에서 검색되고, 찾지 못하면, 링커 검색 경로로 검색된다. 만약 이 옵션이 설정되지 않으면, 호스트의 /etc/ld.so.conf가 사용되고, 그러면 프로그램이 호스트의 라이브러리에 연결될 수 있으며 이는 우리의 목적에 부합하지 않는다.

패키지를 컴파일한다:

```
make
```

패키지를 설치한다:

```
make install
```

이제 다음 장의 링커 “재조정” 단계를 위한 준비를 한다:

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
```

Make 인자들의 의미:

-C ld clean
ld 하위 디렉토리에서 컴파일된 모든 파일을 제거한다.

-C ld LIB_PATH=/usr/lib:/lib
ld 하위 디렉토리의 모든 것을 재빌드한다. 명령줄에 LIB_PATH Makefile 변수를 지정하면 임시 도구의 기본 값을 무시하고 올바른 최종 경로를 가리킬 수 있다. 이 변수의 값이 링커의 기본 라이브러리 검색 경로이다. 이 준비사항은 다음 장에서 쓰인다.

이 패키지에 대한 자세한 정보는 6.18.2절. “Binutils의 내용.”에서 찾을 수 있다.

5.10. GCC-9.2.0 - 2단계

GCC 패키지는 C, C++ 컴파일러가 들어있는 GNU 컴파일러 컬렉션을 포함하고 있다.

예상 빌드 시간: 13 SBU
필요 디스크 공간: 3.7 GB

5.10.1. GCC 설치

우리의 첫 번째 GCC 빌드는 내부 시스템 헤더 몇 개 설치했다. 이 중 하나인 limits.h에는, 보통 해당 시스템의 limits.h 헤더, 즉 /tools/include/limits.h가 포함된다. 그러나 GCC의 첫 빌드 때는 /tools/include/limits.h가 존재하지 않았으므로, GCC가 설치한 내부 헤더는 일부에 불과한 독립적 파일이며 시스템 헤더의 확장 기능이 들어있지 않다. 이는 임시 libc를 빌드하는 데는 적합했지만, 지금 GCC를 빌드하는 데는 전체 내부 헤더가 필요하다. 일반적인 상황에서 GCC 빌드 시스템이 수행하는 것과 동일한 명령을 사용해서 내부 헤더의 전체 버전을 만든다:

```
cat gcc/limitx.h gcc/glimits.h gcc/limity.h > ₩
  `dirname $($LFS_TGT-gcc -print-libgcc-file-name)`/include-fixed/limits.h
```

다시 한 번, /tools에 설치된 것을 사용하도록 GCC의 기본 동적 링커 위치를 변경하라.

```
for file in gcc/config/{linux,i386/linux{,64}}.h
do
  cp -uv $file{,.orig}
  sed -e 's@/lib₩(64₩)₩?₩(32₩)₩?/ld@/tools&@g' ₩
      -e 's@/usr@/tools@g' $file.orig > $file
  echo '
#undef STANDARD_STARTFILE_PREFIX_1
#undef STANDARD_STARTFILE_PREFIX_2
#define STANDARD_STARTFILE_PREFIX_1 "/tools/lib/"
#define STANDARD_STARTFILE_PREFIX_2 ""' >> $file
  touch $file.orig
done
```

x86_64에서 빌드하는 경우, 64비트 라이브러리의 기본 디렉토리 이름을 “lib”로 변경하라:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' ₩
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

GCC의 첫 번째 빌드와 마찬가지로 GMP, MPFR 및 MPC 패키지가 필요하다. tarbal의 압축을 풀고 필요한 디렉토리 이름으로 변경하라:

```
tar -xf ../mpfr-4.0.2.tar.xz
mv -v mpfr-4.0.2 mpfr
tar -xf ../gmp-6.2.0.tar.xz
mv -v gmp-6.2.0 gmp
tar -xf ../mpc-1.1.0.tar.gz
mv -v mpc-1.1.0 mpc
```

이제 Glibc-2.31에서 나온 문제를 해결한다:

```
sed -e '1161 s|^|//|' ₩
    -i libsanitizer/sanitizer_common/sanitizer_platform_limits_posix.cc
```

다시 별도의 빌드 디렉토리를 생성한다:

```
mkdir -v build
cd       build
```

GCC 빌드를 시작하기 전에, 기본 최적화 플래그를 덮어쓰는 환경 변수들을 반드시 해제하라.

GCC 컴파일을 준비한다:

```
CC=$LFS_TGT-gcc                         ₩
CXX=$LFS_TGT-g++                        ₩
AR=$LFS_TGT-ar                          ₩
RANLIB=$LFS_TGT-ranlib                  ₩
../configure                            ₩
    --prefix=/tools                     ₩
    --with-local-prefix=/tools          ₩
    --with-native-system-header-dir=/tools/include ₩
    --enable-languages=c,c++            ₩
    --disable-libstdcxx-pch             ₩
    --disable-multilib                  ₩
    --disable-bootstrap                 ₩
    --disable-libgomp
```

Configure 옵션들의 의미:

--enable-languages=c,c++
C와 C++ 컴파일러가 모두 빌드되도록 한다.

--disable-libstdcxx-pch
libstdc++를 위해 사전 컴파일된 헤더(PCH)를 빌드하지 않도록 한다. 디스크 공간을 많이 차지하며, 우리에게 쓸모가 없다.

--disable-bootstrap
네이티브 GCC 빌드는 "부트스트랩" 빌드가 기본값이다. 이것은 단순히 GCC를 컴파일하는 것이 아니라 여러 번 컴파일한다. 1차 컴파일 결과를 활용해 2차 빌드를 한 뒤, 3차 컴파일을 한다. 두 번째와 세 번째 반복을 거쳐 제스스로 완전히 복제가 가능한 지 비교한다. 이를 통해 정확하게 컴파일됐는지의 여부도 알 수 있다. 그러나 LFS 빌드 과정은 매번 부트스트랩할 필요 없이 견고한 컴파일러를 제공할 것이다.

패키지를 컴파일한다:

```
make
```

패키지를 설치한다:

```
make install
```

마무리 작업으로 심볼릭 링크를 만든다. 많은 프로그램들과 스크립트들은 gcc대신 cc를 실행함으로써 GNU C 컴파일러가 항상 설치되어있지 않은 모든 종류의 UNIX 시스템에서도 사용할 수 있게 범용성을 유지한다. cc를 사용하면 시스템 관리자가 어떤 C 컴파일러를 설치할 지 자유롭게 결정하도록 할 수 있다:

```
ln -sv gcc /tools/bin/cc
```

경고

여기서 잠깐 멈추고 새로운 툴체인의 기본 기능(컴파일 및 링크)이 기대한 대로 잘 작동하는지 확인하는 것이 우선이다. 다음 명령으로 온전성 검사를 수행하라:

```
echo 'int main(){}' > dummy.c
cc dummy.c
readelf -l a.out | grep ': /tools'
```

모든 것이 제대로 작동하면 오류가 없어야 하며 마지막 명령의 출력은 다음과 같을 것이다:

```
[Requesting program interpreter: /tools/lib64/ld-linux-x86-64.so.2]
```

32비트 시스템에서는 동적 링커가 /tools/lib/ld-linux.so.2임을 참고하라.

만약 출력이 위와 같이 표시되지 않거나 출력이 전혀 없다면, 무언가 잘못된 것이다. 이전 단계로 돌아가 어디서 문제가 생겼는지 파악하고 수정하라. 이 문제는 계속 진행하기 전에 반드시 해결해야 한다. 먼저 cc 대신 gcc를 사용하여 온전성 검사를 다시 실행하라. 만약 잘 된다면, /tools/bin/cc가 누락된 것이다. 앞서 언급한 심볼릭 링크를 생성하라. 그런 다음 PATH가 올바른지 확인하라. echo $PATH를 실행했을 때 결과 맨 앞에 /tools/bin가 나오면 올바른 것이다. PATH가 잘못됐다면 lfs 유저로 로그인하지 않았거나 4.4절. “환경 구축.”에서 문제가 발생했다는 의미이다.

모두 잘 됐다면, 테스트 파일을 정리하라:

```
rm -v dummy.c a.out
```

이 패키지에 대한 자세한 정보는 6.25.2절. “GCC의 내용.”에서 찾을 수 있다.

5.11. Tcl-8.6.10

Tcl 패키지는 도구 명령 언어(Tool Command Language)를 포함하고 있다.

예상 빌드 시간: 0.9 SBU
필요 디스크 공간: 72 MB

5.11.1. Tcl 설치

이 패키지와 다음 2개(Expect와 DejaGNU)는 GCC, Binutils 그리고 다른 패키지의 테스트 스위트 실행을 지원하기 위해 설치된다. 테스트 목적으로 3개의 패키지를 설치하는 것은 과도해 보일 수 있지만, 가장 중요한 도구가 제대로 작동하고 있다는 것을 확인하는 것은 필수는 아니더라도 매우 안심할 수 있는 일이다. 이 장에서 테스트 스위트를 실행하지 않더라도(필수 사항이 아님), 6장의 테스트 스위트를 실행하기 위해 필요하다.

여기서 사용되는 Tcl 패키지는 LFS 테스트를 실행하는 데 필요한 최소 버전이라는 점에 유의하라. 전체 패키지는 BLFS Tcl 절차를 참조하라.

Tcl 컴파일을 준비한다:

```
cd unix
./configure --prefix=/tools
```

패키지를 빌드한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. Tcl 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
TZ=UTC make test
```

Tcl 테스트 스위트는 완전히 파악되지 않은 호스트 환경에서 실패할 수도 있다. 따라서 테스트 스위트가 실패해도 놀랄 일이 아니며, 중요하지도 않다. TZ=UTC 매개변수는 표준 시간대를 협정 세계시(UTC)로 설정하지만, 테스트 스위트를 실행하는 동안에만 적용된다. 이렇게 하면 시계 테스트가 올바르게 수행된다. TZ 환경변수에 대한 자세한 내용은 7장에 수록되어 있다.

패키지를 설치한다:

```
make install
```

디버깅 기호를 나중에 지울 수 있도록 설치된 라이브러리를 쓰기 가능으로 설정하라:

```
chmod -v u+w /tools/lib/libtcl8.6.so
```

Tcl의 헤더를 설치한다. 다음 패키지인 Expect를 빌드하는데 필요하다.

```
make install-private-headers
```

이제 필요한 심볼릭 링크를 만든다:

```
ln -sv tclsh8.6 /tools/bin/tclsh
```

5.11.2. Tcl의 내용

설치된 프로그램들: tclsh(tclsh8.6에 링크됨)과 tclsh8.6
설치된 라이브러리: libtcl8.6.so, libtclstub8.6.a

요약

tclsh8.6	Tcl 명령 셸
tclsh	8.6로의 링크
libtcl8.6.so	Tcl 라이브러리
libtclstub8.6.a	Tcl Stub 라이브러리

5.12. Expect-5.45.4

Expect 패키지에는 다른 대화형 프로그램과 스크립트 대화를 수행하기 위한 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 4.0 MB

5.12.1. Expect 설치

먼저 Expect의 configure 스크립트가 호스트 시스템에서 찾을 수 있는 /usr/local/bin/stty 대신 /bin/stty를 사용하도록 강제 설정한다. 이를 통해 테스트 스위트 도구는 툴체인의 최종 빌드에도 그대로 사용할 수 있다:

```
cp -v configure{,.orig}
sed 's:/usr/local/bin:/bin:' configure.orig > configure
```

이제 Expect 컴파일을 준비한다:

```
./configure --prefix=/tools       ₩
            --with-tcl=/tools/lib ₩
            --with-tclinclude=/tools/include
```

Configure 옵션들의 의미:

--with-tcl=/tools/lib
이 옵션을 통해 configure 스크립트가 호스트 시스템에서 기존 것을 찾는 대신 임시 도구 위치에서 Tcl을 찾을 수 있다.

--with-tclinclude=/tools/include
Expect가 Tcl의 내부 헤더를 어디서 찾을지 확실하게 지정한다. 이 옵션을 사용하면 Tcl의 헤더 위치를 자동으로 검색하지 않으므로 configure에 실패하는 상황을 방지할 수 있다.

패키지를 빌드한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make test
```

우리 통제 하에 있지 않은 호스트 상황에서는 Expect 테스트 스위트가 실패할 수 있음을 참고하라. 따라서 이 테스트 스위트가 실패하더라도 놀랄 일이 아니며 중요하지도 않다.

패키지를 설치한다:

```
make SCRIPTS="" install
```

Make 매개 변수의 의미:

SCRIPTS=""
필요하지 않은 추가적인 Expect 스크립트들의 설치를 방지한다.

5.12.2. Expect의 내용

설치된 프로그램들: expect
설치된 라이브러리: libexpect-5.45.so

요약

expect	스크립트를 통해 다른 대화형 프로그램과 통신
libexpect-5.45.so	Expect를 Tcl 확장으로, 또는 C, C++로부터 직접(Tcl없이) 사용할 수 있는 함수들을 포함

5.13. DejaGNU-1.6.2

DejaGNU 패키지는 다른 프로그램들을 테스트하기 위한 프레임워크를 담고 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 3.2 MB

5.13.1. DejaGNU 설치

DejaGNU 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 빌드하고 설치한다:

```
make install
```

다음으로 결과를 테스트한다:

```
make check
```

5.13.2. DejaGNU의 내용

설치된 프로그램: runtest

요약

runtest 적합한 expect 셸을 찾아서 DejaGNU을 실행하는 래퍼(wrapper) 스크립트

5.14. M4-1.4.18

M4 패키지에는 매크로 처리기가 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 20 MB

5.14.1. M4 설치

우선 glibc-2.28에서 알려진 몇 가지 수정 사항을 적용한다:

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

M4 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 M4 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.16.2절. “M4의 내용.”에서 찾을 수 있다.

5.15. Ncurses-6.2

Ncurses 패키지는 문자 화면의 터미널 독립적 처리를 위한 라이브러리를 포함하고 있다.

예상 빌드 시간: 0.6 SBU
필요 디스크 공간: 41 MB

5.15.1. Ncurses 설치

우선 첫 구성 중에 gawk가 검색될 수 있도록 한다:

```
sed -i s/mawk// configure
```

Ncurses 컴파일을 준비한다:

```
./configure --prefix=/tools ₩
            --with-shared   ₩
            --without-debug ₩
            --without-ada   ₩
            --enable-widec  ₩
            --enable-overwrite
```

Configure 옵션들의 의미:

--without-ada
Ncurses가 Ada 컴파일러를 지원하지 않도록 한다. Ada 컴파일러가 호스트에 있을지 몰라도 일단 chroot 환경에 들어가면 사용할 수 없다.

--enable-overwrite
/tools/include/ncurses 대신 /tools/include에 헤더 파일들을 설치하도록 한다. 이를 통해 다른 패키지들도 Ncurses 헤더들을 성공적으로 찾을 수 있다.

--enable-widec
일반 문자 라이브러리(libncurses.so.6.2) 대신 와이드 문자 라이브러리(libncursesw.so.6.2)를 설치하도록 한다. 이런 와이드 문자 라이브러리는 멀티바이트 및 기존의 8비트 언어에서 모두 사용할 수 있지만, 일반 라이브러리는 8비트 언어에서만 제대로 작동한다. 와이드 문자와 일반 문자 라이브러리는 소스에서는 호환되지만 바이너리 호환은 되지 않는다.

패키지를 컴파일한다:

```
make
```

이 패키지는 테스트 스위트가 있지만, 패키지를 설치한 후에만 실행할 수 있다. 테스트들은 test/ 디렉토리에 있다. 자세한 내용은 해당 디렉토리의 README 파일을 참조하라.

패키지를 설치한다:

```
make install
ln -s libncursesw.so /tools/lib/libncurses.so
```

이 패키지에 대한 자세한 정보는 6.27.2절. “Ncurses의 내용.”에서 찾을 수 있다.

5.16. Bash-5.0

Bash 패키지에는 Bourne-Again SHell이 포함되어 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 67 MB

5.16.1. Bash 설치

Bash 컴파일을 준비한다:

```
./configure --prefix=/tools --without-bash-malloc
```

Configure 옵션들의 의미:

--without-bash-malloc
세그먼테이션 오류를 일으키는 것으로 알려진 Bash의 메모리 할당(malloc) 기능을 비활성화한다. 이 옵션을 끄면 Bash는 좀 더 안정적인 Glibc의 malloc 함수를 사용할 것이다.

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Bash 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make tests
```

패키지를 설치한다:

```
make install
```

셸로 sh를 사용하도록 링크를 생성한다:

```
ln -sv bash /tools/bin/sh
```

이 패키지에 대한 자세한 정보는 6.35.2절. “Bash의 내용.”에서 찾을 수 있다.

5.17. Bison-3.5.2

Bison 패키지에는 파서 생성기가 포함되어 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 43 MB

5.17.1. Bison 설치

Bison 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.32.2절. “Bison의 내용.”에서 찾을 수 있다.

5.18. Bzip2-1.0.8

Bzip2 패키지에는 파일 압축 및 압축 해제를 위한 프로그램이 포함되어 있다. bzip2로 텍스트 파일을 압축하면 기존 gzip보다 압축률이 훨씬 높다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 6.4 MB

5.18.1. Bzip2 설치

Bzip2 패키지에는 configure 스크립트가 포함되어 있지 않다. 공유 라이브러리를 위한 것과, 정적 라이브러리를 위한 Makefile이 두 개 있다. 둘 모두 필요하기 때문에 두 단계에 걸쳐 컴파일한다. 우선 공유 라이브러리이다:

```
make -f Makefile-libbz2_so
make clean
```

Make 매개 변수의 의미:

-f Makefile-libbz2_so
: Bzip2가 다른 Makefile 파일로 빌드되도록 한다. 이 경우엔 libbz2.so 동적 라이브러리를 생성하고 Bzip2 유틸리티를 링크하는 Makefile-libbz2_so 파일을 사용한다.

패키지를 컴파일하고 테스트한다:

```
make
```

패키지를 설치한다:

```
make PREFIX=/tools install
cp -v bzip2-shared /tools/bin/bzip2
cp -av libbz2.so* /tools/lib
ln -sv libbz2.so.1.0 /tools/lib/libbz2.so
```

이 패키지에 대한 자세한 정보는 6.12.2절. “Bzip2의 내용.”에서 찾을 수 있다.

5.19. Coreutils-8.31

Coreutils 패키지는 기본적인 시스템 특성을 보여주고 설정하기 위한 유틸리티를 포함하고 있다.

예상 빌드 시간: 0.7 SBU
필요 디스크 공간: 157 MB

5.19.1. Coreutils 설치

Coreutils 컴파일을 준비한다:

```
./configure --prefix=/tools --enable-install-program=hostname
```

Configure 옵션들의 의미:

--enable-install-program=hostname
hostname 바이너리를 빌드하고 설치한다 - 기본적으로 비활성화되어있지만 Perl 테스트 스위트에 필요하다.

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Coreutils 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make RUN_EXPENSIVE_TESTS=yes check
```

RUN_EXPENSIVE_TESTS=yes 매개 변수는, 일반적으로 리눅스에서는 문제가 되지 않지만 일부 플랫폼에서 비교적 비싸다고 간주되는(CPU 전원 및 메모리 사용 측면에서) 몇 가지 추가 테스트를 실행하도록 테스트 스위트에 지시한다.

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.54.2절. “Coreutils의 내용.”에서 찾을 수 있다.

5.20. Diffutils-3.7

Diffutils 패키지는 파일 또는 디렉토리 간의 차이를 보여주는 프로그램을 포함하고 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 26 MB

5.20.1. Diffutils 설치

Diffutils 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Diffutils 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.56.2절. “Diffutils의 내용.”에서 찾을 수 있다.

5.21. File-5.38

File 패키지에는 지정된 파일 또는 파일의 유형을 파악하는 유틸리티가 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 20 MB

5.21.1. File 설치

File 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 File 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.14.2절. “File의 내용.”에서 찾을 수 있다.

5.22. Findutils-4.7.0

Findutils 패키지는 파일을 찾는 프로그램을 포함하고 있다. 디렉토리 트리를 재귀적으로 검색하고 데이터베이스를 작성, 유지관리 및 검색(보통은 재귀 검색보다 빠르지만 데이터베이스가 최근에 업데이트되지 않았다면 신뢰할 수 없다)하기 위한 프로그램이다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 39 MB

5.22.1. Findutils 설치

Findutils 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Findutils 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.58.2절. “Findutils의 내용.”에서 찾을 수 있다.

5.23. Gawk-5.0.1

Gawk 패키지에는 텍스트 파일을 조작하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 46 MB

5.23.1. Gawk 설치

Gawk 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Gawk 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.57.2절. “Gawk의 내용.”에서 찾을 수 있다.

5.24. Gettext-0.20.1

Gettext 패키지에는 다국어화와 현지화를 위한 유틸리티가 포함되어 있다. 이를 통해 프로그램을 현지 언어를 지원(Native Language Support, NLS)하도록 컴파일할 수 있어 사용자의 모국어로 메시지를 출력할 수 있다.

예상 빌드 시간: 1.6 SBU
필요 디스크 공간: 300 MB

5.24.1. Gettext 설치

우리의 임시 도구 세트를 위해선 Gettext로부터 세 개의 프로그램만 설치하면 된다.

Gettext 컴파일을 준비한다:

```
./configure --disable-shared
```

Configure 옵션들의 의미:

--disable-shared
지금은 어떤 공유 Gettext 라이브러리도 설치할 필요가 없으므로 라이브러리를 빌드하지 않는다.

패키지를 컴파일한다:

```
make
```

환경이 제한적이기 때문에 이 단계에서는 테스트 스위트를 실행하지 않는 것이 좋다.

msgfmt, msgmerge 그리고 xgettext 프로그램을 설치한다:

```
cp -v gettext-tools/src/{msgfmt,msgmerge,xgettext} /tools/bin
```

이 패키지에 대한 자세한 정보는 6.47.2절. “Gettext의 내용.”에서 찾을 수 있다.

5.25. Grep-3.4

Grep 패키지는 파일 내용을 검색하는 프로그램을 포함하고 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 25 MB

5.25.1. Grep 설치

Grep 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Grep 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.34.2절. “Grep의 내용.”에서 찾을 수 있다.

5.26. Gzip-1.10

Gzip 패키지에는 파일을 압축하고 해제하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 10 MB

5.26.1. Gzip 설치

Gzip 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Gzip 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.62.2절. “Gzip의 내용.”에서 찾을 수 있다.

5.27. Make-4.3

Make 패키지는 패키지를 컴파일하기 위한 프로그램을 포함하고 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 16 MB

5.27.1. Make 설치

Make 컴파일을 준비한다:

```
./configure --prefix=/tools --without-guile
```

Configure 옵션들의 의미:

--without-guile
Make-4.3가 Guile 라이브러리(호스트 시스템에 존재할 수도 있지만 다음 장의 chroot 환경에서는 사용할 수 없다)와 링크되지 않게한다.

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Make 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.67.2절. “Make의 내용.”에서 찾을 수 있다.

5.28. Patch-2.7.6

Patch 패키지에는 일반적으로 diff 프로그램으로 만든 “patch” 파일을 적용하여 파일을 수정하거나 만드는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 13 MB

5.28.1. Patch 설치

Patch 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Patch 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.68.2절. “Patch의 내용.”에서 찾을 수 있다.

5.29. Perl-5.30.1

Perl 패키지에는 Practical Extraction and Report Language가 포함되어 있다.

예상 빌드 시간: 1.5 SBU
필요 디스크 공간: 275 MB

5.29.1. Perl 설치

Perl 컴파일을 준비한다:

```
sh Configure -des -Dprefix=/tools -Dlibs=-lm -Uloclibpth -Ulocincpth
```

Configure 옵션들의 의미:

-des
: 이것은 세 가지 옵션의 조합이다: -d는 모든 항목에 대해 기본값을 사용한다; -e는 모든 작업의 완료를 보장한다; -s는 필수가 아닌 항목들을 출력하지 않는다.

-Uloclibpth와 -Ulocincpth
: Configuration이 호스트 시스템에 존재할 수 있는 로컬 설치 구성 요소를 검색하도록 하는 변수의 정의를 해제한다.

패키지를 빌드한다:

```
make
```

Perl에 테스트 스위트도 있지만 다음 장에서 설치될 때까지 기다리는 것이 나을 것이다.

여기선 몇 개의 유틸리티와 라이브러리만 설치하면 된다:

```
cp -v perl cpan/podlators/scripts/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.30.1
cp -Rv lib/* /tools/lib/perl5/5.30.1
```

이 패키지에 대한 자세한 정보는 6.41.2절. “Perl의 내용.”에서 찾을 수 있다.

5.30. Python-3.8.1

Python 3 패키지는 Python 개발 환경을 포함하고 있다. 객체 지향 프로그래밍, 스크립트 작성, 대형 프로그램 프로토타입 제작 또는 응용 프로그램 자체를 개발하는 데 유용하다.

예상 빌드 시간: 1.3 SBU
필요 디스크 공간: 409 MB

5.30.1. Python 설치

참고

이름이 “python”으로 시작하는 두 개의 패키지 파일이 있다. 추출할 대상은 Python-3.8.1.tar.xz(첫 번째 글자가 대문자임을 참고)이다.

이 패키지는 우선 파이썬 인터프리터를 빌드한 다음 몇 가지 표준 파이썬 모듈을 만든다. 모듈 빌드를 위한 메인 스크립트는 파이썬으로 작성되며, 호스트 /usr/include와 /usr/lib 디렉토리로 하드코드된 경로를 사용한다. 그 곳으로부터 쓰이는 것을 막기 위해 다음을 실행하라:

```
sed -i '/def add_multiarch_paths/a ₩        return' setup.py
```

Python 컴파일을 준비한다:

```
./configure --prefix=/tools --without-ensurepip
```

Configure 옵션들의 의미:

--without-ensurepip
이 단계에서 필요하지 않은 파이썬 패키지 설치 프로그램을 비활성화 한다.

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 테스트 스위트에는 TK와 X 윈도우가 필요하며 지금은 실행할 수 없다.

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.51.2절. “Python 3의 내용.”에서 찾을 수 있다.

5.31. Sed-4.8

Sed 패키지는 스트림 에디터를 포함하고 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 21 MB

5.31.1. Sed 설치

Sed 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Sed 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.29.2절. “Sed의 내용.”에서 찾을 수 있다.

5.32. Tar-1.32

Tar 패키지에는 아카이빙(archiving) 프로그램을 포함하고 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 38 MB

5.32.1. Tar 설치

Tar 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Tar 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.70.2절. “Tar의 내용.”에서 찾을 수 있다.

5.33. Texinfo-6.7

Texinfo 패키지에는 info 페이지를 읽고, 쓰고, 변환하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 104 MB

5.33.1. Texinfo 설치

Texinfo 컴파일을 준비한다:

```
./configure --prefix=/tools
```

참고

Configure 과정에서 TestXS_la-TestXS.lo에 대한 오류를 출력하는 테스트가 있다. 이것은 LFS와 관련이 없으며 무시해야 한다.

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Texinfo 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.71.2절. “Texinfo의 내용.”에서 찾을 수 있다.

5.34. Xz-5.2.4

Xz 패키지는 파일을 압축하고 압축을 푸는 프로그램을 포함하고 있다. lzma 알고리즘과 새로운 xz 압축 포맷에 대한 기능을 제공한다. 텍스트 파일을 xz로 압축하면 기존의 gzip이나 bzip2보다 압축률이 더 높다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 18 MB

5.34.1. Xz 설치

Xz 컴파일을 준비한다:

```
./configure --prefix=/tools
```

패키지를 컴파일한다:

```
make
```

이제 컴파일이 끝났다. 앞에서 논의한 바와 같이, 이 장의 임시 도구에 대해 테스트 스위트를 반드시 실행해야 하는 것은 아니다. 그래도 Xz 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

이 패키지에 대한 자세한 정보는 6.13.2절. “Xz의 내용.”에서 찾을 수 있다.

5.35. 스트리핑(Stripping)

이 절의 내용은 선택 사항이지만 LFS 파티션이 다소 작을 경우, 불필요한 항목을 제거할 수 있다는 점을 배울 수 있다는 점에서 유익하다. 지금까지 빌드된 실행 파일과 라이브러리에는 약 70 MB의 불필요한 디버깅 심볼이 포함되어 있다. 다음 명령으로 이러한 심볼을 제거하라:

```
strip --strip-debug /tools/lib/*
/usr/bin/strip --strip-unneeded /tools/{,s}bin/*
```

이 명령들은 파일 형식을 인식하지 못하고 여러 파일을 건너뛴다. 이 중 대부분은 바이너리가 아닌 스크립트이다. 시스템 strip 명령을 사용해서 /tools의 strip 바이너리도 포함시킨다.

--strip-unneeded를 라이브러리에는 사용하지 않는다는 점에 주의하라. 그렇지 않으면 정적 라이브러리들은 파괴될 것이고 툴체인 패키지 전체를 다시 빌드해야 한다.

공간을 더 확보하려면 문서를 삭제한다:

```
rm -rf /tools/{,share}/{info,man,doc}
```

불필요한 파일들을 삭제한다:

```
find /tools/{lib,libexec} -name ₩*.la -delete
```

이 때, 다음 단계에서 Glibc와 Gcc를 빌드하고 설치하는 데 사용할 수 있도록 $LFS에 최소 3GB의 여유 공간이 있어야 한다. 만약 Glibc를 빌드하고 설치할 수 있으면, 나머지도 빌드, 설치할 수 있다.

5.36. 소유권 변경

참고

이 책의 나머지 부분에 있는 명령들은 더 이상 lfs 유저가 아닌 root 유저로 로그인한 상태에서 실행해야 한다. 또, $LFS가 root의 환경에 설정되어 있는지 다시 확인하라.

현재 $LFS/tools 디렉토리는 호스트 시스템에만 존재하는 유저인 lfs가 소유하고 있다. $LFS/tools 디렉토리를 그대로 두면 해당 계정이 없는 유저 ID가 파일을 소유하게 된다. 이것은 나중에 동일한 유저 ID로 사용자 계정이 만들어질 수도 있고 그러면 $LFS/tools 디렉토리와 모든 파일을 소유하게 되어 그 안의 파일들이 악의적인 조작에 노출될 수 있기 때문에 위험하다.

이런 문제를 피하려면, 나중에 /etc/passwd 파일을 생성할 때 lfs 유저를 새 LFS 시스템에 추가하고, 호스트 시스템과 동일한 유저 및 그룹 ID를 할당하도록 주의하면 된다. 그래도 더 나은 방법은, 다음 명령으로 $LFS/tools 디렉토리의 소유권을 root 유저로 변경하는 것이다:

```
chown -R root:root $LFS/tools
```

LFS 시스템이 완성되고 나면 $LFS/tools 디렉토리를 삭제할 수 있지만, 같은 버전의 LFS 시스템을 추가로 더 만들기 위해 보관할 수도 있다. $LFS/tools를 어떻게 백업할지는 각자 선택에 달렸다.

경고

추후에 LFS 시스템을 빌드하는 데 사용하기 위해 임시 도구들을 보관하려면 지금 백업하라. 6장 이후의 명령들은 현재 그 도구들을 변경해서 향후 빌드에는 사용할 수 없게 한다.

부 III. LFS 시스템 빌드

6장. 기본 시스템 소프트웨어 설치

6.1. 도입

이 장에서는 빌드 구역에 들어가 LFS 시스템 구축을 본격적으로 시작한다. 즉 우리는 임시 미니 리눅스 시스템으로 chroot 하고 몇 가지 최종 준비를 한 다음 패키지 설치를 시작할 것이다.

이 소프트웨어 설치는 간단하다. 많은 설치 지침들을 일반화시키고 간결하게 작성할 수 있었지만, 우리는 독자가 실수할 가능성을 최소화하기 위해 모든 패키지에 대한 전체 과정을 설명하기로 했다. 리눅스 시스템의 작동 원리를 파악하기 위한 핵심은 각 패키지가 무엇에 사용되며 이 패키지가 여러분(또는 시스템)에게 왜 필요한지 아는 것이다.

우리는 최적화하는 것을 권장하지 않는다. 그렇게 하면 프로그램이 약간 더 빨리 실행되도록 할 수 있지만, 프로그램을 실행할 때 문제를 일으키거나 컴파일에 어려움을 겪을 수도 있다. 최적화를 할 때 패키지를 컴파일하는 데 실패하면 최적화 없이 컴파일을 시도해서 문제가 해결되는지 확인하라. 최적화를 해서 패키지를 컴파일하는 데 성공하더라도 코드와 빌드 툴 간의 복잡한 상호 작용으로 인해 패키지가 잘못 컴파일되었을 위험이 있다. 또한 책에 명시되지 않은 값을 사용한 -march와 -mtune 옵션은 테스트되지 않았다. 이로 인해 툴체인 패키지들(Binutils, GCC 및 Glibc)에 문제가 발생할 수 있다. 컴파일러 최적화를 사용할 때 얻을 수 있는 작은 이득보다 리스크가 더 크다. LFS 처음 사용자는 맞춤형 최적화 없이 구축할 것을 권장한다. 그래도 시스템은 여전히 매우 빠르게 작동하며 동시에 안정적일 것이다.

이 장의 패키지 설치 순서를 엄격히 지켜서 어떤 프로그램도 뜻하지 않게 /tools를 참조하지 않도록 해야 한다. 같은 이유로, 여러 패키지들을 동시에 컴파일하지 않도록 하라. 병렬 컴파일을 하면 시간(특히 듀얼-CPU 컴퓨터)이 절약될 순 있지만 프로그램이 /tools에 대한 내부 참조를 가져, 해당 디렉토리가 제거되고 나면 프로그램이 작동하지 않을 수 있다.

설치 설명에 앞서 각 설치 페이지에는 패키지에 대한 간략한 설명, 패키지에 포함된 내용, 빌드에 걸리는 대략적인 시간, 이 빌드 프로세스 동안 필요한 디스크 공간 등의 정보가 제공된다. 설치 지침에 따라 패키지가 설치하는 프로그램 및 라이브러리 목록(이들에 대한 간략한 설명과 함께)이 있다.

참고

SBU 값과 필요한 디스크 공간은 6장의 적용 가능한 모든 패키지의 테스트 스위트 데이터를 포함한다.

6.1.1. 라이브러리에 대해

일반적으로 LFS 편집자들은 정적 라이브러리의 빌드와 설치를 만류한다. 대부분의 정적 라이브러리의 원래 목적은 현대 리눅스 시스템에서 쓸모가 없어졌다. 게다가 정적 라이브러리를 프로그램과 링크하는 것은 해로울 수가 있다. 보안 문제를 해결하기 위해 라이브러리를 업데이트 해야하는 경우, 정적 라이브러리를 사용하는 모든 프로그램들을 새 라이브러리에 다시 링크해야 한다. 정적 라이브러리를 사용하는지는 항상 알 수 있는 것이 아니기 때문에, 관련 프로그램(또는 링크하는 데 필요한 절차)을 파악할 수조차 없을 수도 있다.

6장에서 우리는 대부분의 정적 라이브러리를 삭제하거나 설치를 비활성화한다. 보통은 configure에 --disable-static 옵션을 전달해서 이뤄진다. 그 외엔 대안이 필요하다. 특히 glibc와 gcc를 비롯한 일부 패키지는 일반적인 빌드 과정에서 정적 라이브러리 사용이 필수적이다.

라이브러리에 대한 더 자세한 논의는 BLFS 책의 라이브러리: 정적 or 공유?를 참고하라.

6.2. 가상 커널 파일 시스템 준비

커널에 의해 내보내진 다양한 파일 시스템들은 커널 자체와 통신하는 데 사용된다. 이러한 파일 시스템은 디스크 공간이 사용되지 않는다는 점에서 가상적이라 할 수 있다. 파일 시스템의 내용들은 메모리에 상주한다.

파일 시스템을 마운트할 디렉토리를 만드는 것부터 시작한다:

```
mkdir -pv $LFS/{dev,proc,sys,run}
```

6.2.1. 초기 장치 노드 생성

커널이 시스템을 부팅할 때 몇 개의 장치 노드, 특히 console과 null 장치가 있어야 한다. 장치 노드는 udevd가 시작되기 전에, 그리고 리눅스가 init=/bin/bash로 시작될 때 사용할 수 있도록 하드 디스크에 생성되어야 한다. 다음 명령을 실행하여 장치를 생성하라:

```
mknod -m 600 $LFS/dev/console c 5 1
mknod -m 666 $LFS/dev/null c 1 3
```

6.2.2. /dev 마운팅과 설정

/dev 디렉토리에 장치를 들이는 권장되는 방법은 /dev 디렉토리에 가상 파일 시스템(예: tmpfs)을 마운트하고, 장치가 감지되거나 액세스될 때 해당 가상 파일 시스템에 동적으로 생성되도록 하는 것이다. 장치 생성은 일반적으로 부팅 절차 중 Udev에 의해 이뤄진다. 지금 이 새로운 시스템은 아직 Udev가 없고 부팅되지도 않았기 때문에 수동으로 마운트를 하고 /dev를 채워야 한다. 이를 위해 호스트 시스템의 /dev 디렉토리를 바인딩한다. 바인딩 마운트란 디렉토리나 마운트 지점의 미러를 다른 위치에 만들 수 있는 특별한 유형의 마운트를 뜻한다. 이 작업을 수행하려면 다음 명령을 사용하라:

```
mount -v --bind /dev $LFS/dev
```

6.2.3. 가상 커널 파일 시스템 마운팅

이제 나머지 가상 커널 파일 시스템을 마운트한다:

```
mount -vt devpts devpts $LFS/dev/pts -o gid=5,mode=620
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

devpts에 대한 마운트 옵션들의 의미:

gid=5
: devpts가 생성한 모드 장치 노드들을 그룹 ID 5가 소유한다. 이것은 우리가 나중에 tty 그룹에 사용할 ID이다. 호스트 시스템이 그 tty 그룹에 다른 ID를 사용할지도 모르기 때문에 이름 대신 그룹 ID를 사용한다.

mode=0620
: devpts가 생성한 모든 장치 노드를 0620 모드(사용자는 읽기/쓰기 가능, 그룹은 쓰기 가능)로 설정한다. 위의 옵션과 함께, devpts가 grantpt()의 요구 사항을 만족하는 장치 노드를 생성하도록 한다. 다시말해 기본적으로 설치되지 않는 Glibc pt_chown helper 바이너리가 필요하지 않다는 의미이다.

어떤 호스트 시스템에서는, /dev/shm이 /run/shm에 대한 심볼릭 링크이다. /run tmpfs은 위에 마운트되었으므로 디렉토리만 작성하면 된다.

```
if [ -h $LFS/dev/shm ]; then
  mkdir -pv $LFS/$(readlink $LFS/dev/shm)
fi
```

6.3. 패키지 관리

패키지 관리를 LFS 책에서 다뤄달라는 요청이 자주 온다. 패키지 관리자를 사용하면 파일 설치를 추적하여 삭제나 업그레이드를 편리하게 할 수 있다. 바이너리와 라이브러리 파일들 뿐만 아니라, 패키지 관리자도 구성 파일들의 설치를 관리한다. 궁금해지기 전에 말해두자면, 안된다—이 절에서는 특정 패키지 관리자에 대해 이야기하거나 추천하지 않을 것이다. 이 절의 내용은 더 인기 있는 테크닉과 그 작동 원리들에 대한 정리이다. 당신을 위한 완벽한 패키지 관리자는 이런 기술들 가운데 하나일수도 둘 이상을 조합한 것일 수도 있다. 이 절에서는 패키지를 업그레이드 할 때 발생할 수 있는 문제에 대해 간략히 설명한다.

LFS나 BLFS에서 패키지 관리자를 언급하지 않는 몇 가지 이유는 다음과 같다:

- 패키지 관리를 다루는 것은 이 책들의 목표에서 벗어난다—리눅스 시스템을 구축하는 방법을 가르치는 것.
- 패키지 관리를 위한 여러 솔루션이 있으며 각각 장점과 단점이 있다. 모든 독자를 만족시키는 것을 찾는 것은 어려운 일이다.

패키지 관리 주제에 대해 몇 가지 힌트가 작성되어있다. 힌트 프로젝트에 방문하여 해당 프로젝트 중 필요한 것이 있는지 확인해보라.

6.3.1. 업그레이드에 관해

패키지 관리자를 사용하면 최신 버전이 배포되었을 때 쉽게 업그레이드할 수 있다. 일반적으로 LFS와 BLFS 책의 지침들을 통해 새로운 버전으로 업그레이드할 수 있다. 다음은 특히 작동 중인 시스템에서, 패키지를 업그레이드할 때 알아야 할 몇 가지 사항이다.

- 만약 Glibc를 새로운 버전(예: glibc-2.19에서 glibc-2.20로)으로 업그레이드해야 한다면, LFS를 재빌드하는 것이 더 안전하다. 모든 패키지를 의존성 순서에 따라 다시 빌드할 수는 있지만, 권장하지는 않는다.
- 공유 라이브러리를 포함하는 패키지가 업데이트되었다면, 또 라이브러리의 이름이 변경되었다면 그 라이브러리에 동적으로 링크된 모든 패키지를 다시 컴파일하여 새 라이브러리와 링크해야 한다(패키지 버전과 라이브러리 이름 사이에는 상관 관계가 없다는 점에 유의). 예를 들어 libfoo.so.1라는 이름의 공유 라이브러리를 설치하는 패키지 foo-1.2.3이 있다고 하자. 이 패키지를 libfoo.so.2라는 이름의 공유 라이브러리를 설치하는 최신 버전의 foo-1.2.4로 업그레이드한다고 가정하자. 이 때 libfoo.so.1에 동적으로 링크된 모든 패키지는 libfoo.so.2에 링크되도록 다시 컴파일해야 한다. 종속 패키지가 다시 컴파일될 때까지 이전 라이브러리를 제거하지 않도록 주의하라.

6.3.2. 패키지 관리 기술들

다음은 몇 가지 일반적인 패키지 관리 기법이다. 패키지 관리자를 결정하기 전에, 다양한 기술, 특히 각 기법의 단점에 대해 조사하라.

6.3.2.1. 모든 것은 내 머릿속에 있다!

그렇다, 이것은 패키지 관리 기법이다. 어떤 사람들은 패키지들을 샅샅이 알고 있고 각 패키지에 의해 어떤 파일들이 설치되는지 알기 때문에 패키지 관리자의 필요성을 느끼지 못한다. 패키지가 변경되면 전체 시스템을 재구축할 계획이어서 패키지 관리가 필요 없는 사용자도 있다.

6.3.2.2. 별도의 디렉토리들에 설치

이는 설치를 관리하는 데 별도의 패키지가 필요 없는 단순한 패키지 관리법이다. 각 패키지는 별도의 디렉토리에 설치된다. 예를 들어 /usr/pkg/foo-1.1에 foo-1.1 패키지를 설치하고 /usr/pkg/foo에서 /usr/pkg/foo-1.1로 심볼릭 링크를 만든다. 새 버전 foo-1.2를 설치하면, /usr/pkg/foo-1.2에 설치되고 이전 심볼릭링크는 새 버전의 심볼릭링크로 대체된다.

PATH, LD_LIBRARY_PATH, MANPATH, INFOPATH, CPPFLAGS 등의 환경 변수에 /usr/pkg/foo를 추가해야 한다. 패키지가 몇 개 이상으로 많아지면, 이 기법은 관리불능이 된다.

6.3.2.3. Symlink 스타일 패키지 관리

기존 패키지 관리 기법의 변형이다. 각 패키지는 이전 방법과 유사하게 설치된다. 그러나 symlink를 만드는 대신 각 파일이 /usr 계층으로 symlink된다. 이러면 환경 변수를 확장하기 위해 수정할 필요가 없어진다. 사용자가 심볼릭 링크를 생성해서 생성을 자동화할 순 있지만 많은 패키지 관리자가 이 방식을 사용해서 작성되었다. 인기 있는 것들 중에는 Stow, Epkg, Graft, Depot 등이 있다.

실제로는 /usr/pkg 계층에 설치되었지만 패키지가 /usr에 설치된 것으로 인식되도록 설치를 조작해야 한다. 이런 식으로 설치하는 것은 여간 복잡한 일이 아니다. 예를 들어 패키지 libfoo-1.1을 설치한다고 가정하면 다음 명령은 패키지를 제대로 설치하지 못할 수도 있다:

```
./configure --prefix=/usr/pkg/libfoo/1.1
make
make install
```

설치는 되지만 의존 패키지는 예상했던 대로 libfoo에 연결되지 않을 수 있다. libfoo에 링크되는 패키지를 컴파일하면 예상했던 /usr/lib/libfoo.so.1이 아닌 /usr/pkg/libfoo/1.1/lib/libfoo.so.1과 연결된 것을 알 수 있다. 올바른 방법은 DESTDIR 전략을 사용하여 패키지를 가짜로 설치하는 것이다. 이 방식은 이렇게 작동한다:

```
./configure --prefix=/usr
make
make DESTDIR=/usr/pkg/libfoo/1.1 install
```

대부분의 패키지가 이러한 방식을 지원하지만, 그렇지 않은 패키지도 있다. 패키지가 이 방식을 지원하지 않으면 수동으로 설치해야 하거나 문제가 있는 패키지를 /opt에 설치하는 것이 더 낫다.

6.3.2.4. 타임스탬프 기반

이 기법에서는 패키지가 설치되기 전에 파일에 타임스탬프를 찍는다. 설치 후 적절한 옵션과 함께 간단한 find 명령을 사용하면 타임스탬프 파일이 생성된 후 설치된 모든 파일의 로그를 생성할 수 있다. 이 방식을 채택한 패키지 관리자는 install-log이다.

이 방법은 단순하다는 장점이 있지만 두 가지 단점이 있다. 설치 중에 파일이 현재 시간 이외의 타임스탬프와 함께 설치되면 패키지 관리자가 해당 파일을 추적하지 않는다. 더구나 이 방법은 한 번에 하나의 패키지를 설치할 때만 사용할 수 있다. 두 개의 패키지가 서로 다른 두 개의 콘솔에 설치될 때는 로그를 신뢰할 수 없다.

6.3.2.5. 설치 스크립트 추적

이 방식은 설치 스크립트가 수행하는 명령을 기록하는 것이다. 사용할 수 있는 두 가지 기법이 있다:

LD_PRELOAD 환경변수를 설치 전에 미리 로드할 라이브러리를 가리키도록 설정할 수 있다. 설치하는 동안 이 라이브러리는 cp, install, mv 등 다양한 실행 파일에 자신을 첨부하고 파일 시스템을 수정하는 시스템콜을 추적해서 설치 중인 패키지를 추적한다. 이 방식을 사용하려면 모든 실행 파일이 suid나 sgid 비트 없이 동적으로 연결되어야 한다. 라이브러리를 미리 로드하면 설치 중에 원치 않는 부작용이 발생할 수 있다. 따라서 패키지 관리자가 아무것도 깨뜨리지 않고 필요한 모든 파일을 기록하도록 몇 가지 테스트를 하는 것이 좋다.

두 번째 방식은 설치 스크립트를 실행하는 동안 발생하는 모든 시스템콜을 기록하는 strace를 사용하는 것이다.

6.3.2.6. 패키지 아카이브 생성

이 방식에서는 앞서 Symlink 스타일 패키지 관리에서 설명한 대로 패키지 설치를 별도의 트리로 위장한다. 설치가 끝나면 설치된 파일들로 패키지 아카이브를 생성한다. 이 보관 파일은 로컬 컴퓨터나 다른 컴퓨터에 패키지를 설치하는 데 사용할 수 있다.

이 방식은 상용 배포판에서 찾을 수 있는 대부분의 패키지 관리자들이 사용한다. 이 방식을 따르는 패키지 관리자의 예로는 RPM (참고로, Linux Standard Base Specification에 필요), pkg-utils, 데비안의 apt, 그리고 젠투의 Portage system이 있다. LFS 시스템에 이런 스타일의 패키지 관리 방식을 채택하려면 http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt을 참고하라.

의존성 정보를 포함하는 패키지 파일의 생성은 복잡할 뿐더러 LFS의 범위를 벗어난다.

슬랙웨어는 패키지 아카이브에 tar 기반 시스템을 사용한다. 이 시스템은 더 복잡한 패키지 관리자들같이 의도적으로 패키지 의존성을 처리하지 않는다. 슬랙웨어 패키지 관리에 대한 자세한 내용은 http://www.slackbook.org/html/package-management.html을 참고하라.

6.3.2.7. 유저 기반 관리

이 LFS만의 독특한 방식은 Matthias Benkmann이 고안했으며 힌트 프로젝트에서 이용할 수 있다. 이 방식에서 각 패키지는 표준 위치에 별도의 사용자로 설치된다. 패키지에 속하는 파일은 사용자 ID를 확인하여 쉽게 식별할 수 있다. 이 방식의 특징과 단점은 이 절에서 설명하기에 너무 복잡하다. 자세한 내용은 http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt의 힌트를 참고하라.

6.3.3. 여러 시스템에 LFS 배포하기

LFS 시스템의 장점 중 하나는 디스크에서의 위치에 좌우되는 파일이 없다는 것이다. LFS 빌드를 동일한 아키텍쳐의 다른 컴퓨터에 기반 시스템으로 복제하는 것은 간단하다. root 디렉토리(압축되지 않은 기본 LFS 빌드 기준 약 250 MB)가 포함된 LFS 파티션에서 tar를 사용하고, 네트워크나 CD-ROM을 통해 새 시스템으로 복사하고 확장하는 것으로 복제할 수 있다. 그 후에 몇 개의 구성 파일을 변경해야 할 것이다. 업데이트가 필요할 수 있는 구성 파일로는 /etc/hosts, /etc/fstab, /etc/passwd, /etc/group, /etc/shadow, /etc/ld.so.conf, /etc/sysconfig/rc.site, /etc/sysconfig/network, /etc/sysconfig/ifconfig.eth0 등이 있다.

시스템 하드웨어와 원래 커널 구성의 차이에 따라 새로운 시스템을 위한 커스텀 커널을 빌드해야 할 수도 있다.

참고

유사하지만 동일하지 않은 아키텍쳐 간에는 복사할 때 일부 문제가 있다는 보고가 있었다. 예를 들어 인텔 시스템의 명령어 세트는 AMD 프로세서와 동일하지 않으며 일부 프로세서의 이후 버전에는 이전 버전에서 사용할 수 없는 명령이 있을 수 있다.

마지막으로 8.4절. “GRUB을 사용하여 부팅 프로세스 설정”을 통해 새 시스템을 부팅 가능하도록 만들어야 한다.

6.4. Chroot 환경 진입

최종 LFS 시스템 빌드와 설치를 시작하기 위해서 chroot 환경으로 진입해야 할 시간이다. root 유저로, 다음 명령을 사용해서 아직 임시 도구들로만 채워진 곳으로 들어간다:

```
chroot "$LFS" /tools/bin/env -i ₩
   HOME=/root                  ₩
   TERM="$TERM"                ₩
   PS1='(lfs chroot) ₩u:₩w₩$ ' ₩
   PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin ₩
   /tools/bin/bash --login +h
```

env 명령에 붙은 -i 옵션은 chroot 환경의 모든 변수를 지운다. 그 후에, HOME, TERM, PS1, 그리고 PATH 변수만 다시 설정된다. TERM=$TERM 구문은 root 내부의 TERM 변수를 root 외부와 동일한 값으로 설정한다. vim과 less 같은 프로그램이 제대로 작동하려면 이 변수가 필요하다. CFLAGS나 CXXFLAGS같은 다른 변수가 필요하다면, 여기서 다시 설정하면 된다.

여기서부터는 모든 작업이 LFS 파일 시스템으로 제한되기 때문에 LFS 변수를 더 이상 사용할 필요가 없다. Bash 셸은 이제 $LFS를 root(/) 디렉토리로 여기기 때문이다.

/tools/bin이 PATH의 마지막에 있다는 점에 유의하라. 이는 일단 최종 버전이 설치되면 임시 도구는 더 이상 사용되지 않는다는 것을 의미한다. 이는 셸이 실행된 바이너리 파일의 위치를 “기억”하지 않게 해서 이뤄진다. 그러기 위해 +h 옵션을 bash로 전달해서 해싱을 비활성화한다.

bash 프롬프트에서 I have no name!이라고 뜨는 것에 주목하라. 이것은 /etc/passwd 파일이 아직 생성되지 않았기 때문에 정상이다.

참고

이 장의 나머지 부분과 다음 장의 모든 명령을 chroot 환경에서 실행하는 것이 중요하다. 어떤 이유로든 (예: 재부팅) 이 환경을 떠나는 경우 설치를 계속하기 전에 6.2.2절. “/dev 마운팅과 설정”와 6.2.3절. “가상 커널 파일 시스템 마운팅”에 설명된 대로 가상 커널 파일 시스템이 마운트되었는지 확인하고 chroot를 다시 입력한 후 설치를 계속하라.

6.5. 디렉토리 생성

이제 LFS 파일 시스템에 몇 가지 체계를 만들어야 할 때다. 다음 명령을 실행하여 표준 디렉토리 트리를 생성하라:

```
mkdir -pv /{bin,boot,etc/{opt,sysconfig},home,lib/firmware,mnt,opt}
mkdir -pv /{media/{floppy,cdrom},sbin,srv,var}
install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
mkdir -pv /usr/{,local/}{bin,include,lib,sbin,src}
mkdir -pv /usr/{,local/}share/{color,dict,doc,info,locale,man}
mkdir -v  /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -v  /usr/libexec
mkdir -pv /usr/{,local/}share/man/man{1..8}
mkdir -v  /usr/lib/pkgconfig

case $(uname -m) in
 x86_64) mkdir -v /lib64 ;;
esac

mkdir -v /var/{log,mail,spool}
ln -sv /run /var/run
ln -sv /run/lock /var/lock
mkdir -pv /var/{opt,cache,lib/{color,misc,locate},local}
```

디렉토리는 기본적으로 권한 모드 755로 작성되지만 모든 디렉토리에 대해서 바람직한 것은 아니다. 위의 명령에서는 두 가지 변경이 이루어진다—한 가지는 root 유저의 홈 디렉토리이고, 다른 하나는 임시 파일들을 위한 디렉토리이다.

첫 번째 모드 변경은 아무나 /root 디렉토리에 들어갈 수 없도록 한다—일반 유저가 자신의 홈 디렉토리에 하는 것과 동일하다. 두 번째 모드 변경은 모든 유저가 /tmp와 /var/tmp 디렉토리에 쓰기 가능하도록 하지만 이들 디렉토리에서 다른 유저의 파일을 삭제할 수는 없도록 한다. 후자는 1777 비트 마스크에서 가장 높은 비트(1)인 이른바 “sticky 비트”에 의해 금지된다.

6.5.1. FHS 준수 알림

디렉토리 트리는 파일 시스템 계층 표준(Filesystem Hierarchy Standard, FHS)을 기반으로 한다(https://refspecs.linuxfoundation.org/fhs.shtml에서 참고 가능). FHS는 /usr/local/games와 /usr/share/games 같은 일부 디렉토리도 선택적으로 존재할 수 있다고 명시한다. 이 책에선 필요한 디렉토리만 만들겠지만, 이러한 디렉토리들은 자유롭게 만들어도 좋다.

6.6. 필수 파일 및 Symlink 생성

일부 프로그램은 아직 존재하지 않는 프로그램에 대한 고정 경로를 사용한다. 이러한 프로그램들을 만족시키기 위해, 이 장의 절차를 거치면서 소프트웨어를 설치한 후 실제 파일로 대체될 여러 심볼릭 링크를 만든다:

```
ln -sv /tools/bin/{bash,cat,chmod,dd,echo,ln,mkdir,pwd,rm,stty,touch} /bin
ln -sv /tools/bin/{env,install,perl,printf}         /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1}                  /usr/lib
ln -sv /tools/lib/libstdc++.{a,so{,.6}}             /usr/lib

ln -sv bash /bin/sh
```

각 링크의 용도:

/bin/bash
: 많은 bash 스크립트가 /bin/bash를 지정한다.

/bin/cat
: 이 경로명은 Glibc의 configure 스크립트에 하드 코딩되어 있다.

/bin/dd
: dd의 경로는 /usr/bin/libtool 유틸리티에 하드 코딩될 것이다.

/bin/echo
: /bin/echo를 참조하는 Glibc의 테스트 스위트 내 테스트 중 하나를 만족시키기 위함이다.

/usr/bin/env
: 이 경로명은 일부 패키지 빌드 절차에 하드 코딩되어 있다.

/usr/bin/install
: install의 경로가 /usr/lib/bash/Makefile.inc 파일로 하드 코딩될 것이다.

/bin/ln
: ln의 경로는 /usr/lib/perl5/5.30.1/<target-triplet>/Config_heavy.pl 파일에 하드 코딩된다.

/bin/pwd
: 일부 configure 스크립트, 특히 Glibc의 것에는 이 경로명이 하드 코딩되어 있다.

/bin/rm
: rm의 경로는 /usr/lib/perl5/5.30.1/<target-triplet>/Config_heavy.pl 파일에 하드 코딩된다.

/bin/stty
이 경로명은 Expect로 하드 코딩되어 있어서, Binutils와 GCC 테스트 스위트 통과에 필요하다.

/usr/bin/perl
많은 Perl 스크립트가 perl 프로그램에 대해 이 경로를 하드 코딩한다.

/usr/lib/libgcc_s.so{,.1}
Glibc의 pthreads 라이브러리가 작동하기 위해 필요하다.

/usr/lib/libstdc++{,.6}
GMP의 C++ 지원과 더불어 Glibc의 테스트 스위트에서 몇 가지 테스트를 하기 위해 필요하다.

/bin/sh
많은 셸 스크립트가 /bin/sh를 하드 코딩한다.

역사적으로 리눅스는 마운팅된 파일 시스템 목록을 /etc/mtab 파일로 관리한다. 최신 커널은 이 목록을 내부적으로 관리하며 /proc 파일 시스템을 통해 사용자에게 공개한다. /etc/mtab이 존재할 것으로 예상하는 유틸리티들을 만족시키기 위해, 다음 심볼릭 링크를 생성하라:

```
ln -sv /proc/self/mounts /etc/mtab
```

root 유저가 로그인할 수 있게 하고 “root” 이름이 인식되게 하려면 /etc/passwd와 /etc/group 파일에 관련 항목이 있어야 한다.

다음 명령을 실행하여 /etc/passwd 파일을 생성하라:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/dev/null:/bin/false
daemon:x:6:6:Daemon User:/dev/null:/bin/false
messagebus:x:18:18:D-Bus Message Daemon User:/var/run/dbus:/bin/false
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
EOF
```

root에 대한 실제 암호(여기서 사용되는 “x”는 구문표시자일 뿐이다)는 나중에 설정된다.

다음 명령을 실행하여 /etc/group 파일을 생성하라:

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:daemon
sys:x:2:
kmem:x:3:
tape:x:4:
tty:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
input:x:24:
mail:x:34:
kvm:x:61:
wheel:x:97:
nogroup:x:99:
users:x:999:
EOF
```

생성된 그룹들은 어떤 표준에도 속하지 않는다—부분적으로는 이 장의 Udev 구성의 요구사항에 의해, 부분적으로는 기존 리눅스 배포판에 사용된 공통 규약에 의해 만들어진 그룹이다. 그리고 일부 테스트 스위트들은 특정 유저나 그룹을 필요로 한다. 리눅스 기본 규격(Linux Standard Base, LSB, http://www.linuxbase.org에서 참고)은 그룹 ID (GID) 0인 root 그룹 외에, GID가 1인 bin 그룹도 존재하게 둘 것을 권장한다. 다른 모든 그룹명과 GID는 잘 작성된 프로그램이라면 GID 번호에 의존하지 않고 그룹명을 사용하기 때문에 시스템 관리자가 자유롭게 정할 수 있다.

“I have no name!” 프롬프트를 지우려면 새 셸을 시작하라. 5장에서 완전한 Glibc가 설치되었고 /etc/passwd와 /etc/group 파일이 생성되었으므로 이제 유저명과 그룹명을 잘 불러올 것이다:

```
exec /tools/bin/bash --login +h
```

+h 지시문의 사용에 주목하라. 이것은 bash의 내부 경로 해싱을 비활성화 한다. 이 명령이 없으면 bash는 바이너리를 실행한 경로를 기억할 것이다. 새로 컴파일된 바이너리 파일이 설치되는 즉시 사용될 수 있도록 이 장에서는 +h 지시문을 사용할 것이다.

login, agetty 및 init 프로그램(및 기타)은 누가, 언제 시스템에 로그인했는지 같은 정보를 기록하기 위해 여러 로그 파일을 사용한다. 그러나 이런 프로그램들은 로그 파일이 이미 존재하지 않으면 로그를 남기지 않는다. 로그 파일을 초기화하고 적절한 권한을 부여하라:

```
touch /var/log/{btmp,lastlog,faillog,wtmp}
chgrp -v utmp /var/log/lastlog
chmod -v 664  /var/log/lastlog
chmod -v 600  /var/log/btmp
```

/var/log/wtmp 파일은 모든 로그인과 로그아웃을 기록한다. /var/log/lastlog 파일은 각 유저가 마지막으로 로그인한 시간을 기록한다. /var/log/faillog 파일은 유저의 로그인 실패 정보를 기록한다. /var/log/btmp 파일은 잘못된 로그인 시도를 기록한다.

참고

/run/utmp 파일은 현재 로그인한 유저를 기록한다. 이 파일은 부팅 스크립트에서 동적으로 생성된다.

6.7. 리눅스-5.5.3 API 헤더

리눅스-5.5.3.tar.xz의 리눅스 API 헤더는 Glibc에 쓰이는 커널 API를 담고있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 1 GB

6.7.1. 리눅스 API 헤더 설치

리눅스 커널은 시스템의 C 라이브러리(LFS의 Glibc)가 사용할 수 있도록 API(응용프로그램 인터페이스)를 공개해야 한다. 이것은 리눅스 커널 소스 tarball로 배포되는 다양한 C 헤더 파일을 검사하는 것으로 이뤄진다.

이전 작업으로 인해 오래된 파일이나 의존성이 남아있지는 않은지 확인하라:

```
make mrproper
```

이제 소스에서 사용자가 볼 수 있는 커널 헤더를 추출하라. 권장되는 make 대상 “headers_install”은 /tools에서 사용할 수 없는 rsync가 필요하기 때문에 사용할 수 없다. 헤더를 먼저 ./usr에 배치한 다음, 커널 개발자가 사용하는 일부 파일을 제거하고 파일을 최종 위치로 복사한다.

```
make headers
find usr/include -name '.*' -delete
rm usr/include/Makefile
cp -rv usr/include/* /usr/include
```

6.7.2. 리눅스 API 헤더의 내용

설치된 헤더들: /usr/include/asm/*.h, /usr/include/asm-generic/*.h, /usr/include/drm/*.h, /usr/include/linux/*.h, /usr/include/misc/*.h, /usr/include/mtd/*.h, /usr/include/rdma/*.h, /usr/include/scsi/*.h, /usr/include/sound/*.h, /usr/include/video/*.h, and /usr/include/xen/*.h
설치된 디렉토리들: /usr/include/asm, /usr/include/asm-generic, /usr/include/drm, /usr/include/linux, /usr/include/misc, /usr/include/mtd, /usr/include/rdma, /usr/include/scsi, /usr/include/sound, /usr/include/video, and /usr/include/xen

요약

/usr/include/asm/*.h	리눅스 API ASM 헤더
/usr/include/asm-generic/*.h	리눅스 API ASM 범용 헤더
/usr/include/drm/*.h	리눅스 API DRM 헤더
/usr/include/linux/*.h	리눅스 API 리눅스 헤더
/usr/include/misc/*.h	리눅스 API 기타 헤더
/usr/include/mtd/*.h	리눅스 API MTD 헤더
/usr/include/rdma/*.h	리눅스 API RDMA 헤더
/usr/include/scsi/*.h	리눅스 API SCSI 헤더
/usr/include/sound/*.h	리눅스 API 사운드 헤더
/usr/include/video/*.h	리눅스 API 비디오 헤더
/usr/include/xen/*.h	리눅스 API Xen 헤더

6.8. Man-pages-5.05

The Man-pages 패키지는 2,200개 이상의 man 페이지들을 포함한다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 31 MB

6.8.1. Man-pages 설치

다음을 실행하여 Man-pages을 설치하라:

```
make install
```

6.8.2. Man-pages의 내용

설치된 파일들: 다양한 man 페이지들

요약

man pages C 프로그래밍 언어 함수, 중요한 장치 파일들, 중요한 구성 파일들을 설명

6.9. Glibc-2.31

Glibc 패키지는 메인 C 라이브러리를 포함하고 있다. 이 라이브러리는 메모리 할당, 디렉토리 검색, 파일 열기 및 닫기, 파일 읽기 및 쓰기, 문자열 처리, 패턴 대조, 산술 등의 기본 루틴을 제공한다.

예상 빌드 시간: 19 SBU
필요 디스크 공간: 5.5 GB

6.9.1. Glibc 설치

참고

Glibc 빌드 시스템은 컴파일러 설정 파일과 링커가 여전히 /tools를 가리키고 있더라도 독립적이며 완벽하게 설치될 것이다. 컴파일러 설정들과 링커는 Glibc가 설치되기 전에는 조정할 수 없다. Glibc autoconf 테스트가 실패해서 깔끔한 빌드라는 목표를 이룰 수 없기 때문이다.

어떤 Glibc 프로그램은 FHS를 준수하지 않는 /var/db 디렉토리에 런타임 데이터를 저장한다. 다음 패치를 적용하여 그러한 프로그램의 런타임 데이터가 FHS를 준수하는 위치에 저장되도록 한다:

```
patch -Np1 -i ../glibc-2.31-fhs-1.patch
```

LSB 준수를 위한 심볼릭 링크를 생성한다. 추가로, x86_64의 경우 동적 로더가 올바르게 작동하는 데 필요한 호환성 심볼릭 링크를 생성한다:

```
case $(uname -m) in
  i?86)   ln -sfv ld-linux.so.2 /lib/ld-lsb.so.3
  ;;
  x86_64) ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64
          ln -sfv ../lib/ld-linux-x86-64.so.2 /lib64/ld-lsb-x86-64.so.3
  ;;
esac
```

Glibc 문서에서는 전용 빌드 디렉토리에서 Glibc를 빌드할 것을 권장한다:

```
mkdir -v build
cd       build
```

Glibc 컴파일을 준비한다:

```
CC="gcc -ffile-prefix-map=/tools=/usr" ₩
../configure --prefix=/usr                      ₩
             --disable-werror                   ₩
             --enable-kernel=3.2                ₩
             --enable-stack-protector=strong    ₩
             --with-headers=/usr/include        ₩
             libc_cv_slibdir=/lib
```

옵션들과 새 Configure 매개변수들의 의미:

CC="gcc -ffile-prefix-map=/tools=/usr"
GCC가 /tools 안의 파일에 대한 모든 참조를, 파일들이 /usr에 있는 상태에서 컴파일되는 것처럼 기록하게 한다. 이는 디버깅 기호에 잘못된 경로가 들어가는 것을 방지한다.

--disable-werror
GCC에 전달된 -Werror 옵션을 비활성화한다. 테스트 스위트를 실행하는데 필요하다.

--enable-stack-protector=strong
스택 스매싱 공격과 같은 버퍼 오버플로우를 점검하기 위한 코드를 추가해서 시스템 보안을 향상시킨다.

--with-headers=/usr/include
빌드 시스템이 커널 API 헤더를 어디서 찾을지 지정한다. 기본적으로 이들 헤더는 /tools/include에서 검색된다.

libc_cv_slibdir=/lib
모든 시스템에 대해 올바른 라이브러리를 설정한다. 우리는 lib64를 사용하지 않을 것이다.

패키지를 컴파일한다:

```
make
```

중요

이 절에서 Glibc에 대한 테스트 스위트는 매우 중요하다. 어떤 상황에서도 건너뛰지 않도록 하라.

일반적으로 몇 가지 테스트는 통과하지 못한다. 아래 나열된 테스트 실패는 보통 무시해도 무방하다.

```
case $(uname -m) in
  i?86)   ln -sfnv $PWD/elf/ld-linux.so.2        /lib ;;
  x86_64) ln -sfnv $PWD/elf/ld-linux-x86-64.so.2 /lib ;;
esac
```

참고

위의 심볼릭 링크는 chroot 환경의 빌드 단계에서 테스트를 실행하기 위해 필요하다. 아래 과정에서 덮어쓰인다.

```
make check
```

몇 개의 테스트는 실패할 것이다. Glibc 테스트 스위트는 호스트 시스템에 다소 의존적이다. 다음은 일부 버전의 LFS에서 나타나는 가장 일반적인 문제들이다:

- misc/tst-ttyname 은 LFS chroot 환경에서 실패한다고 알려져 있다.
- inet/tst-idna_name_classify 은 LFS chroot 환경에서 실패한다고 알려져 있다.
- posix/tst-getaddrinfo4와 posix/tst-getaddrinfo5 는 일부 아키텍쳐에서 실패할 수도 있다.
- The nss/tst-nss-files-hosts-multi 테스트는 알려지지 않은 이유로 인해 실패할 수도 있다.
- The rt/tst-cputimer{1,2,3} 테스트는 호스트 시스템 커널에 달려있다. 커널 4.14.91-4.14.96, 4.19.13-4.19.18, 그리고 4.20.0-4.20.5는 이 테스트들에 실패한다고 알려져 있다.
- CPU가 비교적 새로운 인텔/AMD 프로세서가 아닌 시스템에서는 산술 테스트에 실패하는 경우가 있다.

문제되지는 않는 메시지지만, Glibc가 설치되는 도중 /etc/ld.so.conf를 찾을 수 없다는 경고가 출력될 것이다. 다음과 같이 방지하라:

```
touch /etc/ld.so.conf
```

생성된 Makefile을 수정하여 불완전한 LFS 환경에서 실패하는 불필요한 온전성 검사를 건너뛴다:

```
sed '/test-installation/s@$(PERL)@echo not running@' -i ../Makefile
```

패키지를 설치한다:

```
make install
```

nscd를 위한 configuration 파일과 런타임 디렉토리를 설치한다:

```
cp -v ../nscd/nscd.conf /etc/nscd.conf
mkdir -pv /var/cache/nscd
```

다음으로 시스템이 다른 언어로 응답할 수 있도록 로케일을 설치한다. 로케일은 꼭 필요하진 않지만, 이 중 일부가 누락되면 추후에 패키지 테스트 스위트가 중요한 테스트 케이스를 생략할 수도 있다.

각각의 로케일은 localedef 프로그램을 사용하여 설치할 수 있다. 예를 들어, 아래 첫 번째 localedef 명령은 문자셋에 독립적인(charset-independent) /usr/share/i18n/locales/cs_CZ 로케일 정의를 /usr/share/i18n/charmaps/UTF-8.gz 문자셋 정의와 조합하고 그 결과를 /usr/lib/locale/locale-archive 파일에 추가한다. 다음 지침은 테스트에 필요한 최소 로케일 세트를 설치한다:

```
mkdir -pv /usr/lib/locale
localedef -i POSIX -f UTF-8 C.UTF-8 2> /dev/null || true
localedef -i cs_CZ -f UTF-8 cs_CZ.UTF-8
localedef -i de_DE -f UTF-8 de_DE
localedef -i de_DE@euro -f UTF-85 de_DE@euro
localedef -i de_DE -f UTF-8 de_DE.UTF-8
localedef -i el_GR -f ISO-8859-7 el_GR
localedef -i en_GB -f UTF-8 en_GB.UTF-8
localedef -i en_HK -f UTF-8 en_HK
localedef -i en_PH -f UTF-8 en_PH
localedef -i en_US -f UTF-8 en_US
localedef -i en_US -f UTF-8 en_US.UTF-8
localedef -i es_MX -f UTF-8 es_MX
localedef -i fa_IR -f UTF-8 fa_IR
localedef -i fr_FR -f UTF-8 fr_FR
localedef -i fr_FR@euro -f UTF-85 fr_FR@euro
localedef -i fr_FR -f UTF-8 fr_FR.UTF-8
localedef -i it_IT -f UTF-8 it_IT
localedef -i it_IT -f UTF-8 it_IT.UTF-8
localedef -i ja_JP -f EUC-JP ja_JP
localedef -i ja_JP -f SHIFT_JIS ja_JP.SIJS 2> /dev/null || true
localedef -i ja_JP -f UTF-8 ja_JP.UTF-8
localedef -i ru_RU -f KOI8-R ru_RU.KOI8-R
localedef -i ru_RU -f UTF-8 ru_RU.UTF-8
localedef -i tr_TR -f UTF-8 tr_TR.UTF-8
localedef -i zh_CN -f GB18030 zh_CN.GB18030
localedef -i zh_HK -f BIG5-HKSCS zh_HK.BIG5-HKSCS
```

추가로, 당신의 국가, 언어 및 문자 세트에 대한 로케일을 설치하라.

아니면, glibc-2.31/localedata/SUPPORTED 파일에 나열된 모든 로케일(위에 나열된 모든 로케일 포함)을 이 시간 잡아먹는 명령으로 한 번에 설치하라:

```
make localedata/install-locales
```

그런 다음, 필요하다면 localedef 명령을 사용하여 glibc-2.31/localedata/SUPPORTED 파일에 나열되지 않은 로케일을 만들고 설치하라.

참고

Glibc는 이제 다국어화 도메인 이름을 확인할 때 libidn2를 사용한다. 이것은 런타임 의존성이다. 이 기능이 필요하다면 BLFS libidn2 페이지에 libidn2 설치 지침이 있다.

6.9.2. Glibc 구성

6.9.2.1. nsswitch.conf 추가

Glibc 기본값은 네트워크 환경에서 잘 작동하지 않기 때문에 /etc/nsswitch.conf 파일을 생성해야 한다.

다음을 실행하여 /etc/nsswitch.conf 파일을 새로 생성하라:

```
cat > /etc/nsswitch.conf << "EOF"
# Begin /etc/nsswitch.conf

passwd: files
group: files
shadow: files

hosts: files dns
networks: files

protocols: files
services: files
ethers: files
rpc: files

# End /etc/nsswitch.conf
EOF
```

6.9.2.2. 표준 시간대 데이터 추가

다음과 같이 표준 시간대 데이터를 설치하고 설정하라:

```
tar -xf ../../tzdata2019c.tar.gz

ZONEINFO=/usr/share/zoneinfo
mkdir -pv $ZONEINFO/{posix,right}

for tz in etcetera southamerica northamerica europe africa antarctica  ₩
          asia australasia backward pacificnew systemv; do
    zic -L /dev/null   -d $ZONEINFO       ${tz}
    zic -L /dev/null   -d $ZONEINFO/posix ${tz}
    zic -L leapseconds -d $ZONEINFO/right ${tz}
done

cp -v zone.tab zone1970.tab iso3166.tab $ZONEINFO
zic -d $ZONEINFO -p America/New_York
unset ZONEINFO
```

zic 명령들의 의미:

zic -L /dev/null ...
: 윤초가 없는 posix 표준 시간대를 만든다. 이것들을 zoneinfo와 zoneinfo/posix에 모두 넣는 것이 일반적이다. POSIX 표준 시간대를 zoneinfo에 넣어야 한다. 그렇지 않으면 여러 테스트 스위트가 오류를 일으킨다. 디스크 공간이 작은 임베디드 시스템에서 표준 시간대를 업데이트할 의도가 없다면 posix 디렉토리를 사용하지 않음으로써 1.9 MB를 절약할 수 있지만, 일부 응용 프로그램이나 테스트 스위트는 오작동을 일으킬 수 있다.

zic -L leapseconds ...
: 윤초가 포함된 정확한 표준 시간대를 만든다. 디스크 공간이 좁아 표준 시간대를 업데이트하지 않을 임베디드 시스템이나, 정확한 시간이 필요없는 경우는 right 디렉토리를 생략해서 1.9 MB를 절약할 수 있다.

zic ... -p ...
: posixrules 파일을 생성한다. POSIX는 미국 규정에 따라 일광 절약 시간제(또는 섬머타임)가 필요하므로 우리는 뉴욕을 사용한다.

다음 스크립트를 실행하여 로컬 표준 시간대를 정할 수 있다:

```
tzselect
```

지역에 관한 몇 가지 질문에 답하고 나면, 스크립트는 시간대의 이름을 출력할 것이다(예: America/Edmonton). /usr/share/zoneinfo에는 Canada/Eastern이나 EST5EDT처럼 스크립트에 의해 식별되지 않지만 사용할 수 있는 시간대도 있다.

그리고나서 다음을 실행하여 /etc/localtime 파일을 생성하라:

```
ln -sfv /usr/share/zoneinfo/<xxx> /etc/localtime
```

선택한 표준 시간대의 이름으로 <xxx>를 대체하라(예: Asia/Seoul).

6.9.2.3. 동적 로더 구성

기본적으로 동적 로더(/lib/ld-linux.so.2)는 프로그램이 실행될 때 필요한 동적 라이브러리를 /lib와 /usr/lib에서 검색한다. 그러나 라이브러리가 /lib이나 /usr/lib 이외의 디렉토리에 있다면, 동적 로더가 찾을 수 있도록 /etc/ld.so.conf 파일에 라이브러리 위치를 추가해야 한다. 추가 라이브러리를 포함하는 디렉토리는 보편적으로 /usr/local/lib 및 /opt/lib 이 두 디렉토리이므로, 해당 디렉토리들을 동적 로더의 검색 경로에 추가한다.

다음 명령을 실행하여 새 /etc/ld.so.conf 파일을 생성하라:

```
cat > /etc/ld.so.conf << "EOF"
# Begin /etc/ld.so.conf
/usr/local/lib
/opt/lib

EOF
```

원한다면, 동적 로더가 디렉토리를 검색하고 거기서 발견된 파일의 내용을 포함시키도록 할 수도 있다. 보통 이 include 디렉토리의 파일은 포함하려는 라이브러리 경로를 지정하는 하나의 행이다. 이 기능을 추가하려면 다음 명령을 실행하라:

```
cat >> /etc/ld.so.conf << "EOF"
# Add an include directory
include /etc/ld.so.conf.d/*.conf

EOF
mkdir -pv /etc/ld.so.conf.d
```

6.9.3. Glibc의 내용

설치된 프로그램들:	catchsegv, gencat, getconf, getent, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, makedb, mtrace, nscd, pcprofiledump, pldd, sln, sotruss, sprof, tzselect, xtrace, zdump, zic
설치된 라이브러리들:	ld-2.31.so, libBrokenLocale.{a,so}, libSegFault.so, libanl.{a,so}, libc.{a,so}, libc_nonshared.a, libcrypt.{a,so}, libdl.{a,so}, libg.a, libm.{a,so}, libmcheck.a, libmemusage.so, libmvec.{a,so}, libnsl.{a,so}, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libpcprofile.so, libpthread.{a,so}, libpthread_nonshared.a, libresolv.{a,so}, librt.{a,so}, libthread_db.so, libutil.{a,so}
설치된 디렉토리들:	/usr/include/arpa, /usr/include/bits, /usr/include/gnu, /usr/include/net, /usr/include/netash, /usr/include/netatalk, /usr/include/netax25, /usr/include/neteconet, /usr/include/netinet, /usr/include/netipx, /usr/include/netiucv, /usr/include/netpacket, /usr/include/netrom, /usr/include/netrose, /usr/include/nfs, /usr/include/protocols, /usr/include/rpc, /usr/include/sys, /usr/lib/audit, /usr/lib/gconv, /usr/lib/locale, /usr/libexec/getconf, /usr/share/i18n, /usr/share/zoneinfo, /var/cache/nscd, /var/lib/nss_db

요약

catchsegv	프로그램이 세그멘테이션 오류로 종료될 때 스택 기록을 만드는 데 사용
gencat	메시지 카탈로그를 생성

getconf	파일 시스템 특정 변수에 대한 시스템 구성 값 표시
getent	관리 데이터베이스로부터 항목을 수집
iconv	문자 집합 변환
iconvconfig	fastloading iconv 모듈 구성 파일 생성
ldconfig	동적 링커 런타임 바인딩 구성
ldd	지정된 각각의 프로그램이나 공유 라이브러리에 필요한 공유 라이브러리를 출력
lddlibc4	객체 파일을 사용하여 ldd를 보조
locale	현재 로케일에 대한 다양한 정보 출력
localedef	로케일 사양을 컴파일
makedb	텍스트 입력으로 단순 데이터베이스 작성
mtrace	메모리 추적 파일을 읽고 해석하며 사람이 읽을 수 있는 형식으로 표시
nscd	대부분의 보편적인 서비스 요청에 대한 캐시를 제공하는 데몬
pcprofiledump	PC 프로파일링에 의해 생성된 정보 덤프
pldd	프로세스 실행 시 사용되는 동적 공유 개체 나열
sln	정적 링크된 ln 프로그램
sotruss	지정된 명령의 공유 라이브러리 프로시저 호출 추적
sprof	공유 개체 프로파일링 데이터 읽기 및 출력
tzselect	사용자에게 시스템 위치를 묻고 해당 표준 시간대 설명을 출력
xtrace	현재 실행 중인 함수를 출력하여 프로그램 실행 추적
zdump	표준 시간대 덤퍼
zic	표준 시간대 컴파일러
ld-2.31.so	공유 라이브러리 실행 파일용 헬퍼 프로그램
libBrokenLocale	Glibc가 내부적으로 고장난 프로그램(예: 일부 Motif 응용 프로그램들)을 실행하기 위해 사용. 자세한 내용은 glibc-2.31/locale/broken_cur_max.c의 설명을 참고하라
libSegFault	catchsegv에서 사용하는 세그멘테이션 오류 신호 처리기
libanl	비동기식 이름 검색 라이브러리
libc	메인 C 라이브러리
libcrypt	암호 라이브러리
libdl	동적 링크 인터페이스 라이브러리
libg	함수가 없는 더미 라이브러리. 예전엔 g++의 런타임 라이브러리였다
libm	수학 라이브러리
libmcheck	링크되었을 때 메모리 할당 확인 활성화
libmemusage	memusage에서 프로그램의 메모리 사용에 대한 정보를 수집하는 데 사용
libnsl	네트워크 서비스 라이브러리
libnss	호스트 이름, 유저 이름, 그룹 이름, alias, 서비스, 프로토콜 등을 확인하기 위한 기능이 포함된 이름 서비스 스위치 라이브러리(The Name Service Switch libraries)

libpcprofile	실행 파일을 PC 프로필에 사전 로드 가능
libpthread	POSIX 스레드 라이브러리
libresolv	패킷을 생성하고 인터넷 도메인 네임 서버로 전송 및 해석하는 함수 내장
librt	POSIX.1b Realtime Extension에서 지정한 대부분의 인터페이스를 제공하는 함수 내장
libthread_db	다중 스레드 프로그램을 위한 디버거 구축에 유용한 함수 내장
libutil	다양한 Unix 유틸리티에 사용되는 “표준” 함수에 대한 코드 내장

6.10. 툴체인 조정

이제 최종 C 라이브러리가 설치되었으므로, 툴체인(toolchain)을 조정해서 새롭게 컴파일될 프로그램들이 이 새로운 라이브러리들에 연결되도록 해야 한다.

먼저 /tools 링커를 백업하고 5장에서 만든 조정된 링커로 교체한다. 또 /tools/$(uname -m)-pc-linux-gnu/bin에 해당 링커에 대한 심볼릭 링크도 만들 것이다:

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(uname -m)-pc-linux-gnu/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(uname -m)-pc-linux-gnu/bin/ld
```

다음으로 GCC spec 파일이 새로운 동적 링커를 참조하도록 수정한다. 단순히 “/tools”의 모든 인스턴스를 삭제하면 동적 링커에 대한 올바른 경로가 남는다. 또한 GCC가 올바른 헤더와 Glibc 시작 파일들을 찾을 수 있도록 spec 파일을 조정한다. 다음 sed 명령을 실행하라:

```
gcc -dumpspecs | sed -e 's@/tools@@g'                   ₩
    -e '/₩*startfile_prefix_spec:/{n;s@.*@/usr/lib/ @}' ₩
    -e '/₩*cpp:/{n;s@$@ -isystem /usr/include@}' >      ₩
    `dirname $(gcc --print-libgcc-file-name)`/specs
```

Spec 파일이 실제로 의도한 대로 수정됐는지 직접 확인하는 것이 좋다.

여기서 조정된 툴체인의 기본 기능(컴파일 및 링크)이 예상대로 작동하는지 확인하는 것이 필수이다. 다음 온전성 검사를 수행하라:

```
echo 'int main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

마지막 명령의 출력은(플랫폼에 따른 동적 링커 이름의 차이는 무관) 다음과 같으며 오류가 없어야 한다:

```
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

64비트 시스템에서 동적 링커의 위치는 /lib이지만, /lib64의 심볼릭 링크를 통해 액세스된다는 점을 참고하라.

참고

32비트 시스템에서 인터프리터는 /lib/ld-linux.so.2여야 한다.

이제 올바른 시작 파일을 사용하도록 설정되었는지 확인하라:

```
grep -o '/usr/lib.*/crt[1in].*succeeded' dummy.log
```

이 명령의 출력은 다음과 같아야 한다:

```
/usr/lib/../lib/crt1.o succeeded
/usr/lib/../lib/crti.o succeeded
/usr/lib/../lib/crtn.o succeeded
```

컴파일러가 올바른 헤더 파일을 검색하는지 확인하라:

```
grep -B1 '^ /usr/include' dummy.log
```

이 명령은 다음과 같은 결과를 출력해야 한다:

```
#include <...> search starts here:
 /usr/include
```

그 다음, 새 링커가 올바른 검색 경로와 함께 사용되고 있는지 확인하라:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |₩n|g'
```

'-linux-gnu' 요소가 있는 경로에 대한 참조는 무시해도 되지만, 나머지 결과는 다음과 같아야 한다:

```
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib")
```

다음으로 올바른 libc를 사용하고 있는지 확인하라:

```
grep "/lib.*/libc.so.6 " dummy.log
```

이 명령의 출력은 다음과 같아야 한다:

```
attempt to open /lib/libc.so.6 succeeded
```

마지막으로 GCC가 올바른 동적 링커를 사용하는지 확인하라:

```
grep found dummy.log
```

이 명령의 출력은 다음과 같아야 한다(플랫폼에 따른 동적 링커 이름의 차이는 무관):

```
found ld-linux-x86-64.so.2 at /lib/ld-linux-x86-64.so.2
```

만약 출력이 위와 같이 나타나지 않거나 출력이 전혀 없다면, 무언가 심각하게 잘못된 것이다. 이전 단계를 조사하고 추적해서 문제가 어디에 있는지 파악하고 수정하라. 가장 유력한 원인은 spec 파일 조정에 이상이 생겼기 때문이다. 어떤 문제든 계속 진행하기 전에 해결해야 할 것이다.

모든 것이 올바르게 작동하면 테스트 파일을 정리하라:

```
rm -v dummy.c a.out dummy.log
```

6.11. Zlib-1.2.11

Zlib 패키지는 일부 프로그램에서 사용하는 압축 및 압축 해제 루틴을 포함하고 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 5.1 MB

6.11.1. Zlib 설치

Zlib 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 다음을 수행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

공유 라이브러리를 /lib로 이동해야 하므로, /usr/lib의 .so 파일을 다시 생성해야 한다:

```
mv -v /usr/lib/libz.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libz.so) /usr/lib/libz.so
```

6.11.2. Zlib의 내용

설치된 라이브러리들: libz.{a,so}

요약

libz 일부 프로그램에서 사용하는 압축 및 압축 해제 기능 포함

6.12. Bzip2-1.0.8

Bzip2 패키지에는 파일 압축 및 압축 해제를 위한 프로그램이 포함되어 있다. bzip2로 텍스트 파일을 압축하면 기존 gzip보다 압축률이 훨씬 높다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 7.7 MB

6.12.1. Bzip2 설치

이 패키지의 문서를 설치하는 패치를 적용한다:

```
patch -Np1 -i ../bzip2-1.0.8-install_docs-1.patch
```

다음 명령은 심볼릭 링크를 상대적으로 생성한다:

```
sed -i 's@₩(ln -s -f ₩)$(PREFIX)/bin/@₩1@' Makefile
```

Man 문서들을 올바른 위치에 설치하도록 만든다:

```
sed -i "s@(PREFIX)/man@(PREFIX)/share/man@g" Makefile
```

Bzip2 컴파일을 준비한다:

```
make -f Makefile-libbz2_so
make clean
```

Make 매개변수의 의미:

-f Makefile-libbz2_so
: 다른 Makefile 파일을 사용해서 Bzip2를 빌드한다. 여기서 Makefile-libbz2_so 파일은 동적 libbz2.so 라이브러리를 생성하고 거기에 Bzip2 유틸리티를 링크한다.

패키지를 컴파일하고 테스트한다:

```
make
```

패키지를 설치한다:

```
make PREFIX=/usr install
```

공유 bzip2 바이너리를 /bin 디렉토리에 설치하고 필요한 심볼릭 링크를 만든 후 정리한다:

```
cp -v bzip2-shared /bin/bzip2
cp -av libbz2.so* /lib
ln -sv ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm -v /usr/bin/{bunzip2,bzcat,bzip2}
ln -sv bzip2 /bin/bunzip2
ln -sv bzip2 /bin/bzcat
```

6.12.2. Bzip2의 내용

설치된 프로그램들: bunzip2 (bzip2로 링크), bzcat (bzip2로 링크), bzcmp (bzdiff로 링크), bzdiff, bzegrep (bzgrep로 링크), bzfgrep (bzgrep로 링크), bzgrep, bzip2, bzip2recover, bzless (bzmore로 링크), bzmore
설치된 라이브러리들: libbz2.{a,so}
설치된 디렉토리: /usr/share/doc/bzip2-1.0.8

요약

bunzip2	bzip으로 압축된 파일을 압축 해제
bzcat	표준 출력으로 압축 해제
bzcmp	bzip으로 압축된 파일에 대해 cmp를 실행
bzdiff	bzip으로 압축된 파일에 대해 diff를 실행
bzegrep	bzip으로 압축된 파일에 대해 egrep를 실행
bzfgrep	bzip으로 압축된 파일에 대해 fgrep를 실행
bzgrep	bzip으로 압축된 파일에 대해 grep를 실행
bzip2	허프만(Huffman) 코딩과 Borrows-Wheeler 블록 정렬 텍스트 압축 알고리즘을 사용하여 파일을 압축; gzip과 같은 “Lempel-Ziv” 알고리즘을 사용하는 기존의 압축기보다 압축률이 더 우수함
bzip2recover	손상된 bzip 압축 파일에서 데이터 복구 시도
bzless	bzip으로 압축된 파일에 대해 less를 실행
bzmore	bzip으로 압축된 파일에 대해 more를 실행
libbz2	Burrows-Wheeler 알고리즘을 사용하여 무손실 블록 정렬 데이터 압축을 구현하는 라이브러리

6.13. Xz-5.2.4

Xz 패키지는 파일을 압축하고 압축을 푸는 프로그램을 포함하고 있다. lzma 알고리즘과 새로운 xz 압축 포맷에 대한 기능을 제공한다. 텍스트 파일을 xz로 압축하면 기존의 gzip이나 bzip2보다 압축률이 더 높다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 16 MB

6.13.1. Xz 설치

Xz 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --docdir=/usr/share/doc/xz-5.2.4
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 다음을 실행하라:

```
make check
```

패키지를 설치하고 모든 필수 파일을 올바른 디렉토리로 옮긴다:

```
make install
mv -v   /usr/bin/{lzma,unlzma,lzcat,xz,unxz,xzcat} /bin
mv -v /usr/lib/liblzma.so.* /lib
ln -svf ../../lib/$(readlink /usr/lib/liblzma.so) /usr/lib/liblzma.so
```

6.13.2. Xz의 내용

설치된 프로그램들: lzcat (xz로 링크), lzcmp (xzdiff로 링크), lzdiff (xzdiff로 링크), lzegrep (xzgrep로 링크), lzfgrep (xzgrep로 링크), lzgrep (xzgrep로 링크), lzless (xzless로 링크), lzma (xz로 링크), lzmadec, lzmainfo, lzmore (xzmore로 링크), unlzma (xz로 링크), unxz (xz로 링크), xz, xzcat (xz로 링크), xzcmp (xzdiff로 링크), xzdec, xzdiff, xzegrep (xzgrep로 링크), xzfgrep (xzgrep로 링크), xzgrep, xzless, and xzmore
설치된 라이브러리들: liblzma.so
설치된 디렉토리들: /usr/include/lzma와 /usr/share/doc/xz-5.2.4

요약

lzcat	표준 출력으로 압축 해제
lzcmp	LZMA 압축 파일에 대해 cmp 실행
lzdiff	LZMA 압축 파일에 대해 diff 실행
lzegrep	LZMA 압축 파일에 대해 egrep 실행
lzfgrep	LZMA 압축 파일에 대해 fgrep 실행
lzgrep	LZMA 압축 파일에 대해 grep 실행
lzless	LZMA 압축 파일에 대해 less 실행

lzma	LZMA 포맷으로 파일을 압축 또는 압축 해제
lzmadec	LZMA 압축 파일을 위한 작고 빠른 디코더
lzmainfo	LZMA 압축 파일 헤더에 저장된 정보를 출력
lzmore	LZMA 압축 파일에 대해 more 실행
unlzma	LZMA 포맷을 사용하여 파일 압축 해제
unxz	XZ 포맷을 사용하여 파일 압축 해제
xz	XZ 포맷을 사용하여 파일 압축 또는 압축 해제
xzcat	표준 출력으로 압축 해제
xzcmp	LZMA 압축 파일에 대해 cmp 실행
xzdec	XZ 압축 파일을 위한 작고 빠른 디코더
xzdiff	LZMA 압축 파일에 대해 diff 실행
xzegrep	LZMA 압축 파일에 대해 egrep 실행
xzfgrep	LZMA 압축 파일에 대해 fgrep 실행
xzgrep	LZMA 압축 파일에 대해 grep 실행
xzless	LZMA 압축 파일에 대해 less 실행
xzmore	LZMA 압축 파일에 대해 more 실행
liblzma	Lempel-Ziv-Markov 체인 알고리즘을 사용하여 무손실 블록 정렬 데이터 압축을 구현하는 라이브러리

6.14. File-5.38

File 패키지에는 지정된 파일 또는 파일의 유형을 파악하는 유틸리티가 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 20 MB

6.14.1. File 설치

File 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 다음을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.14.2. File의 내용

설치된 프로그램: file
설치된 라이브러리: libmagic.so

요약

file	지정된 각 파일에 대해 분류를 시도; 여러 테스트를 수행함—파일 시스템 테스트, 매직 넘버 테스트, 언어 테스트
libmagic	file 프로그램에서 사용하는 매직 넘버 인식 루틴을 포함

6.15. Readline-8.0

Readline 패키지는 명령줄 편집 및 기록 기능을 제공하는 라이브러리 모음이다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 15 MB

6.15.1. Readline 설치

Readline을 재설치하면 이전 라이브러리가 <libraryname>.old로 변경된다. 일반적으로는 문제가 되지 않지만, 경우에 따라 ldconfig의 링크 버그를 유발할 수 있다. 이는 다음 두 개의 sed 명령을 실행해서 방지할 수 있다:

```
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
```

Readline 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --docdir=/usr/share/doc/readline-8.0
```

패키지를 컴파일한다:

```
make SHLIB_LIBS="-L/tools/lib -lncursesw"
```

Make 옵션의 의미:

SHLIB_LIBS="-L/tools/lib -lncursesw"
Readline을 libncursesw 라이브러리에 강제로 링크한다.

이 패키지는 테스트 스위트가 포함되어있지 않다.

패키지를 설치한다:

```
make SHLIB_LIBS="-L/tools/lib -lncursesw" install
```

이제 동적 라이브러리를 보다 적절한 위치로 이동하고 권한 및 심볼릭 링크를 수정하라:

```
mv -v /usr/lib/lib{readline,history}.so.* /lib
chmod -v u+w /lib/lib{readline,history}.so.*
ln -sfv ../../lib/$(readlink /usr/lib/libreadline.so) /usr/lib/libreadline.so
ln -sfv ../../lib/$(readlink /usr/lib/libhistory.so ) /usr/lib/libhistory.so
```

원한다면 문서를 설치하라:

```
install -v -m644 doc/*.{ps,pdf,html,dvi} /usr/share/doc/readline-8.0
```

6.15.2. Readline의 내용

설치된 라이브러리들: libhistory.so와 libreadline.so
설치된 디렉토리들: /usr/include/readline과 /usr/share/doc/readline-8.0

요약

libhistory history의 기록을 불러오기 위한 일관된 유저 인터페이스 제공

libreadline	프로그램의 대화형 세션에 입력된 텍스트를 조작하기 위한 명령어 세트를 제공

6.16. M4-1.4.18

M4 패키지에는 매크로 처리기가 포함되어 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 33 MB

6.16.1. M4 설치

우선 glibc-2.28에 필요한 몇 가지 수정을 한다:

```
sed -i 's/IO_ftrylockfile/IO_EOF_SEEN/' lib/*.c
echo "#define _IO_IN_BACKUP 0x100" >> lib/stdio-impl.h
```

M4 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 다음 명령을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.16.2. M4의 내용

설치된 프로그램: m4

요약

m4 지정된 파일을 복사하는 동시에 파일에 포함된 매크로를 확장 [이러한 매크로들은 내장되어 있거나 사용자 정의되어 있으며 얼마든지 인수를 받을 수 있다. m4는 매크로 확장 외에도 명명된 파일을 포함, 유닉스 명령 실행, 정수 산술 계산, 텍스트 조작, 재귀 등을 위한 기능이 내장되어 있다. m4 프로그램은 컴파일러의 프론트 엔드나 그 자체로 매크로 프로세서로 사용할 수 있다.]

6.17. Bc-2.5.3

Bc 패키지는 임의의 정밀 숫자 처리 언어를 포함하고 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 2.9 MB

6.17.1. Bc 설치

Bc 컴파일을 준비한다:

```
PREFIX=/usr CC=gcc CFLAGS="-std=c99" ./configure.sh -G -O3
```

Configure 옵션들의 의미:

CC=gcc CFLAGS="-std=c99"
사용할 컴파일러 및 C 표준을 지정한다.

-O3
사용할 최적화 단계를 지정한다.

-G
GNU bc가 없으면 작동하지 않는 테스트 스위트의 일부를 생략한다.

패키지를 컴파일한다:

```
make
```

bc를 테스트하려면 실행하라:

```
make test
```

패키지를 설치한다:

```
make install
```

6.17.2. Bc의 내용

설치된 프로그램들: bc와 dc

요약

bc 명령 줄 계산기
dc 후위 표기법 명령 줄 계산기

6.18. Binutils-2.34

Binutils 패키지는 링커, 어셈블러 및 객체 파일을 처리하기 위한 기타 도구를 포함한다.

예상 빌드 시간: 6.7 SBU
필요 디스크 공간: 5.1 GB

6.18.1. Binutils 설치

단순 테스트를 수행하여 PTY가 chroot 환경 내에서 올바르게 작동하는지 확인하라:

```
expect -c "spawn ls"
```

이 명령은 다음을 출력해야 한다:

```
spawn ls
```

만약 출력에 위의 결과 대신 아래 메시지가 포함된 경우, PTY의 올바른 작동을 위한 환경이 설정되지 않은 것이다. 이 문제는 Binutils 및 GCC에 대한 테스트 스위트를 실행하기 전에 해결해야 한다:

```
The system has no more ptys.
Ask your system administrator to create more.
```

이제 테스트를 완료하지 못하게 하는 테스트 하나를 제거하라:

```
sed -i '/@₩tincremental_copy/d' gold/testsuite/Makefile.in
```

Binutils 문서에서는 전용 빌드 디렉토리에서 Binutils를 빌드할 것을 권장한다:

```
mkdir -v build
cd       build
```

Binutils 컴파일을 준비한다:

```
../configure --prefix=/usr       ₩
             --enable-gold       ₩
             --enable-ld=default ₩
             --enable-plugins    ₩
             --enable-shared     ₩
             --disable-werror    ₩
             --enable-64-bit-bfd ₩
             --with-system-zlib
```

Configure 매개변수들의 의미:

--enable-gold
gold 링커를 빌드하고 ld.gold(기본 링커와 함께)로 설치한다.

--enable-ld=default
오리지널 bfd 링커를 빌드해서 ld(기본 링커)와 ld.bfd 두 가지로 설치한다.

--enable-plugins
링커의 플러그인 지원 활성화.

--enable-64-bit-bfd
64비트 지원(워드 사이즈가 더 작은 호스트에서) 활성화. 64비트 시스템에서는 필요하지 않을 수 있지만 해를 끼치진 않는다.

--with-system-zlib
포함된 버전을 빌드하지 않고 설치된 zlib 라이브러리를 사용한다.

패키지를 컴파일한다:

```
make tooldir=/usr
```

Make 매개변수의 의미:

tooldir=/usr
일반적으로 tooldir(실행 파일이 최종적으로 위치할 디렉토리)는 $(exec_prefix)/$(target_alias)로 설정된다. 예를 들어 x86_64 시스템에서는 /usr/x86_64-unknown-linux-gnu로 나올 것이다. 이것은 커스텀 시스템이기 때문에/usr의 대상별 디렉토리는 필요하지 않다. 시스템이 크로스 컴파일(예를 들어, Intel 시스템에서 패키지를 컴파일하여 PowerPC 시스템이 실행할 코드를 생성)을 한다면 $(exec_prefix)/$(target_alias)가 쓰였을 것이다.

중요

이 절의 Binutils 테스트 스위트는 매우 중요하다. 어떤 상황에서도 건너뛰지 않도록 하라.

결과를 테스트하라:

```
make -k check
```

ver_test_pr16504.sh 테스트는 실패하는 것으로 알려져 있다.

패키지를 설치한다:

```
make tooldir=/usr install
```

6.18.2. Binutils의 내용

설치된 프로그램들:	addr2line, ar, as, c++filt, dwp, elfedit, gprof, ld, ld.bfd, ld.gold, nm, objcopy, objdump, ranlib, readelf, size, strings, strip
설치된 라이브러리들:	libbfd.{a,so}와 libopcodes.{a,so}
설치된 디렉토리:	/usr/lib/ldscripts

요약

addr2line	프로그램 주소를 파일 이름 및 줄 번호로 변환; 주소와 실행 파일의 이름을 지정하면 실행 파일의 디버깅 정보를 사용하여 주소와 연결된 원본 파일 및 줄 번호를 결정
ar	아카이브에서 생성, 수정 및 추출
as	gcc의 출력을 오브젝트 파일로 조립하는 어셈블러
c++filt	링커에서 C++와 자바 심볼을 복원하고 함수들이 충돌하지 않고 오버로드되도록 유지하기 위해 사용
dwp	DWARF 패키징 유틸리티
elfedit	ELF 파일의 ELF 헤더 업데이트

gprof	호출 그래프 프로필 데이터 표시
ld	여러 오브젝트 파일과 아카이브 파일을 단일 파일로 조합해서 데이터를 재배치하고 심볼 참조를 연결하는 링커
ld.gold	elf 오브젝트 파일 포맷만 지원하는 ld의 축소 버전
ld.bfd	ld로의 하드 링크
nm	지정된 오브젝트 파일에서 나오는 심볼들을 나열
objcopy	오브젝트 파일의 한 유형을 다른 형식으로 변환
objdump	표시할 특정 정보를 제어하는 옵션과 함께, 지정된 오브젝트 파일에 대한 정보를 표시한다. 표시된 정보는 컴파일 도구를 사용하는 프로그래머에게 유용하다.
ranlib	아카이브의 내용의 색인을 생성해서 아카이브에 저장; 색인은 재배치 가능한 오브젝트 파일들로 이뤄진 아카이브 멤버에 의해 정의된 모든 심볼들을 나열한다.
readelf	ELF 유형 바이너리 파일에 대한 정보 표시
size	지정된 오브젝트 파일들의 섹션 크기 및 총 크기 나열
strings	각 지정된 파일에 대해 지정된 길이(기본값은 4개) 이상의 인쇄 가능한 문자열을 출력한다; 오브젝트 파일의 경우 기본적으로 초기화 및 로드 섹션의 문자열만 출력하고, 다른 파일 형식은 전체 파일을 조사한다
strip	오브젝트 파일에서 심볼 삭제
libbfd	바이너리 파일 디스크립터 라이브러리
libctf	ANSI-C 유형 디버깅 지원 라이브러리
libctf-nobfd	libbfd 기능을 사용하지 않는 libctf 변형
libopcodes	opcode들을 처리하기 위한 라이브러리—프로세서 명령어의 “읽을 수 있는 텍스트” 버전; objdump와 같은 유틸리티 빌드에 사용된다

6.19. GMP-6.2.0

GMP 패키지에는 수학 라이브러리가 포함되어 있다. 정밀 산술에 유용한 함수들을 가지고 있다.

예상 빌드 시간: 1.1 SBU
필요 디스크 공간: 51 MB

6.19.1. GMP 설치

참고

64비트 CPU에서 32비트 x86용으로 빌드하고 있고, 그와 동시에 CFLAGS도 지정했다면, configure 스크립트는 64비트용으로 구성하려다 결국 실패할 것이다. 아래 configure 명령을 실행해서 이 문제를 방지하라:

```
ABI=32 ./configure ...
```

참고

GMP의 기본 설정은 호스트 프로세서에 최적화된 라이브러리를 생성한다. 호스트의 CPU보다 성능이 떨어지는 프로세서에 적합한 라이브러리를 원한다면 다음을 실행하여 일반 라이브러리를 생성할 수 있다:

```
cp -v configfsf.guess config.guess
cp -v configfsf.sub   config.sub
```

GMP 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --enable-cxx     ₩
            --disable-static ₩
            --docdir=/usr/share/doc/gmp-6.2.0
```

Configure 옵션들의 의미:

--enable-cxx
C++ 지원을 활성화한다

--docdir=/usr/share/doc/gmp-6.2.0
문서의 올바른 위치를 지정한다.

패키지를 컴파일하고 HTML 문서를 생성한다:

```
make
make html
```

중요

이 절의 GMP 테스트 스위트는 매우 중요하다. 어떤 상황에서도 건너뛰지 않도록 하라.

결과를 테스트하라:

```
make check 2>&1 | tee gmp-check-log
```

경고

GMP의 코드는 해당 코드를 빌드한 프로세서에 대해 고도로 최적화되어 있다. 드문 일이지만, 프로세서를 감지하는 코드가 시스템 기능을 잘못 식별해서 gmp 라이브러리를 사용하는 테스트나 응용 프로그램으로부터 "Illegal instruction"이라는 메시지와 함께 오류가 발생할 수 있다. 이럴 때는 --build=x86_64-unknown-linux-gnu 옵션과 함께 재구성하고 다시 빌드해야 한다.

테스트 스위트의 190개 테스트가 모두 통과했는지 확인하라. 다음 명령을 실행하여 결과를 확인하라:

```
awk '/# PASS:/{total+=$3} ; END{print total}' gmp-check-log
```

문서와 패키지를 설치한다:

```
make install
make install-html
```

6.19.2. GMP의 내용

설치된 라이브러리들: libgmp.so와 libgmpxx.so
설치된 디렉토리: /usr/share/doc/gmp-6.2.0

요약

libgmp 정밀 산술 함수 포함
libgmpxx C++ 정밀 산술 함수 포함

6.20. MPFR-4.0.2

MPFR 패키지는 다중 정밀 산술 기능을 포함하고 있다.

예상 빌드 시간: 0.8 SBU
필요 디스크 공간: 37 MB

6.20.1. MPFR 설치

MPFR 컴파일을 준비한다:

```
./configure --prefix=/usr        ₩
            --disable-static     ₩
            --enable-thread-safe ₩
            --docdir=/usr/share/doc/mpfr-4.0.2
```

패키지를 컴파일하고 HTML 문서를 생성한다:

```
make
make html
```

중요

이 절의 MPFR 테스트 스위트는 매우 중요하다. 어떤 상황에서도 건너뛰지 않도록 하라.

결과를 테스트하고 모든 테스트가 통과했는지 확인하라:

```
make check
```

패키지와 문서를 설치하라:

```
make install
make install-html
```

6.20.2. MPFR의 내용

설치된 라이브러리: libmpfr.so
설치된 디렉토리: /usr/share/doc/mpfr-4.0.2

요약

libmpfr 다중 정밀 연산 함수를 포함

6.21. MPC-1.1.0

MPC 패키지에는 임의의 높은 정밀도를 갖는 복잡한 숫자의 계산과 정확한 반올림을 위한 라이브러리를 포함하고 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 22 MB

6.21.1. MPC 설치

MPC 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --docdir=/usr/share/doc/mpc-1.1.0
```

패키지를 컴파일하고 HTML 문서를 생성하라:

```
make
make html
```

결과를 테스트하려면, 다음을 실행하라:

```
make check
```

패키지와 문서를 설치하라:

```
make install
make install-html
```

6.21.2. MPC의 내용

설치된 라이브러리: libmpc.so
설치된 디렉토리: /usr/share/doc/mpc-1.1.0

요약

libmpc 복잡한 연산 함수를 포함

6.22. Attr-2.4.48

Attr 패키지에는 파일 시스템 개체의 확장 속성을 관리하는 유틸리티가 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 4.2 MB

6.22.1. Attr 설치

Attr 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --bindir=/bin    ₩
            --disable-static ₩
            --sysconfdir=/etc ₩
            --docdir=/usr/share/doc/attr-2.4.48
```

패키지를 컴파일한다:

```
make
```

테스트는 ext2, ext3 또는 ext4 파일 시스템과 같은 확장 속성을 지원하는 파일 시스템에서 실행되어야 한다. 결과를 테스트하려면 다음을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

공유 라이브러리를 /lib로 이동해야 하므로 /usr/lib의 .so 파일을 다시 만들어야 한다:

```
mv -v /usr/lib/libattr.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libattr.so) /usr/lib/libattr.so
```

6.22.2. Attr의 내용

설치된 프로그램들: attr, getfattr과 setfattr
설치된 라이브러리: libattr.so
설치된 디렉토리들: /usr/include/attr과 /usr/share/doc/attr-2.4.48

요약

attr	파일 시스템 개체에 대한 특성 확장
getfattr	파일 시스템 개체의 확장 특성을 가져옴
setfattr	파일 시스템 개체의 확장 특성 설정
libattr	확장 속성을 조작하는 라이브러리 기능 포함

6.23. Acl-2.2.53

Acl 패키지에는 액세서 제어 목록을 관리하는 유틸리티가 포함되어 있다. 이 유틸리티는 파일 및 디렉토리에 대한 보다 세분화된 액세스 권한을 정의하는 데 사용된다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 6.4 MB

6.23.1. Acl 설치

Acl 컴파일을 준비한다:

```
./configure --prefix=/usr        ₩
            --bindir=/bin        ₩
            --disable-static     ₩
            --libexecdir=/usr/lib ₩
            --docdir=/usr/share/doc/acl-2.2.53
```

패키지를 컴파일한다:

```
make
```

Acl 테스트는 Coreutils가 Acl 라이브러리와 함께 빌드된 후, 액세스 관리를 지원하는 파일 시스템에서 실행되어야 한다. 테스트를 원하면 이 장 뒷부분에서 Coreutils를 빌드한 후 make check를 실행하라.

패키지를 설치한다:

```
make install
```

공유 라이브러리를 /lib로 이동해야 하므로 /usr/lib의 .so 파일을 다시 만들어야 한다:

```
mv -v /usr/lib/libacl.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libacl.so) /usr/lib/libacl.so
```

6.23.2. Acl의 내용

설치된 프로그램들: chacl, getfacl과 setfacl
설치된 라이브러리: libacl.so
설치된 디렉토리들: /usr/include/acl과 /usr/share/doc/acl-2.2.53

요약

chacl	파일 또는 디렉토리의 액세스 제어 목록 변경
getfacl	파일 액세스 제어 목록을 가져옴
setfacl	파일 액세스 제어 목록 설정
libacl	액세스 제어 목록을 조작하는 라이브러리 함수 포함

6.24. Shadow-4.8.1

Shadow 패키지에는 안전한 방법으로 암호를 처리하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 46 MB

6.24.1. Shadow 설치

참고

강력한 암호를 사용하도록 강제하려면, Shadow를 빌드하기 전에 http://www.linuxfromscratch.org/blfs/view/9.1/postlfs/cracklib.html를 참고하여 CrackLib를 설치하라. 그런 다음 아래 configure 명령에 --with-libcrack를 추가하라.

Coreutils가 더 나은 버전을 제공하므로 groups 프로그램과 해당 man 페이지 설치를 비활성화하라. 또, 6.8절. “Man-pages-5.05”에서 이미 설치한 man 페이지 설치를 방지한다:

```
sed -i 's/groups$(EXEEXT) //' src/Makefile.in
find man -name Makefile.in -exec sed -i 's/groups₩.1 / /'   {} ₩;
find man -name Makefile.in -exec sed -i 's/getspnam₩.3 / /' {} ₩;
find man -name Makefile.in -exec sed -i 's/passwd₩.5 / /'   {} ₩;
```

기본 crypt 방식을 사용하는 대신, 암호의 길이를 8자 이상 허용해서 보다 더 안전한 SHA-512 방식을 사용하라. Shadow가 기본적으로 사용하는 유저 메일박스의 위치도 /var/spool/mail에서 /var/mail로 변경해야 한다:

```
sed -i -e 's@#ENCRYPT_METHOD DES@ENCRYPT_METHOD SHA512@' ₩
       -e 's@/var/spool/mail@/var/mail@' etc/login.defs
```

참고

앞에서 Cracklib과 함께 Shadow를 빌드하기로 했다면 다음을 실행하라:

```
sed -i 's@DICTPATH.*@DICTPATH₩t/lib/cracklib/pw_dict@' etc/login.defs
```

useradd로 생성되는 첫 번째 그룹 번호 1000을 약간 수정한다:

```
sed -i 's/1000/999/' etc/useradd
```

Shadow 컴파일을 준비한다:

```
./configure --sysconfdir=/etc --with-group-name-max-length=32
```

Configure 옵션의 의미:

--with-group-name-max-length=32
유저명의 최대 길이는 32자이다. 그룹명의 최대 길이도 같게 한다.

패키지를 컴파일한다:

```
make
```

이 패키지는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

6.24.2. Shadow 설정

이 패키지에는 유저와 그룹을 추가, 수정 및 삭제하고, 유저 및 그룹의 암호를 설정 및 변경하며, 기타 관리 작업을 수행하는 유틸리티가 포함되어 있다. password shadowing의 의미에 대한 자세한 설명은 압축 해제된 소스 트리에 있는 doc/HOWTO 파일을 참고하라. Shadow 지원을 사용할 경우 암호를 확인해야 하는 프로그램들(디스플레이 관리자, FTP 프로그램들, pop3 데몬 등)은 반드시 Shadow와 호환되어야 함을 명심하라. 다시 말해, 그 프로그램들은 Shadow로 처리된 암호로 작동이 가능해야 한다.

Shadowed 암호를 활성화하려면, 다음 명령을 실행하라:

```
pwconv
```

Shadowed 그룹 암호를 활성화하려면:

```
grpconv
```

useradd 유틸리티에 대한 Shadow의 기본 설정은 설명이 필요한 몇 가지 주의사항이 있다. 우선 useradd 유틸리티의 기본 동작은 유저와 동일한 이름의 그룹을 만드는 것이다. 기본적으로 유저 ID (UID)와 그룹 ID (GID) 번호는 1000으로 시작한다. 즉, 매개 변수를 useradd에 전달하지 않으면, 각 유저가 시스템에서 유일한 한 그룹의 구성원이 된다는 의미이다. 이런 방식이 마음에 들지 않는다면 -g 매개 변수를 useradd에 전달해야 한다. 기본 매개 변수는 /etc/default/useradd 파일에 저장된다. 따라서 동작 방식을 바꾸려면 이 파일에서 두 개의 매개 변수를 수정해야 한다.

/etc/default/useradd 매개 변수 설명

GROUP=1000
: 이 매개 변수는 /etc/group 파일에 사용되는 그룹 번호의 시작을 설정한다. 원하는 대로 수정할 수 있다. useradd는 UID나 GID를 재사용하지 않는다는 점에 주의하라. 이 매개 변수에 이미 사용된 번호를 대입하면 그 숫자 다음으로 사용 가능한 번호를 사용한다. 또 -g 매개 변수 없이 useradd를 처음 사용할 때 시스템에 그룹 1000이 없으면, 터미널에 useradd: unknown GID 1000이라는 메시지가 표시된다는 점을 참고하라. 그룹 번호 1000이 사용될 것이기 때문에 이 메시지는 무시해도 된다.

CREATE_MAIL_SPOOL=yes
: 이 매개 변수는 useradd가 새로 만든 유저의 메일박스 파일을 생성하게 한다. useradd는 이 파일의 그룹 소유권을 0660 권한을 가진 mail 그룹에 할당한다. 이러한 메일박스 파일을 useradd가 생성하지 않게 하려면 다음 명령을 실행하라:

```
sed -i 's/yes/no/' /etc/default/useradd
```

6.24.3. root 비밀번호 설정

root 유저의 암호를 정하고 다음을 실행하여 설정하라:

```
passwd root
```

6.24.4. Shadow의 내용

설치된 프로그램들: chage, chfn, chgpasswd, chpasswd, chsh, expiry, faillog, gpasswd, groupadd, groupdel, groupmems, groupmod, grpck, grpconv, grpunconv, lastlog, login, logoutd, newgidmap, newgrp, newuidmap, newusers, nologin, passwd, pwck, pwconv, pwunconv, sg (newgrp로 링크), su, useradd, userdel, usermod, vigr (vipw로 링크)와 vipw

설치된 디렉토리: /etc/default

요약

chage	비밀번호를 변경해야하는 날짜의 최대 기간을 변경하는 데 사용
chfn	사용자의 전체 이름 및 기타 정보를 변경하는 데 사용
chgpasswd	배치 모드에서 그룹 암호를 업데이트하는 데 사용
chpasswd	배치 모드에서 유저 암호를 업데이트하는 데 사용
chsh	사용자의 기본 로그인 셸을 변경하는 데 사용
expiry	현재 암호 만료 정책 확인 및 적용
faillog	로그인 실패 로그를 검사, 계정이 차단되기 전의 최대 실패 횟수를 설정, 실패 횟수를 재설정하는 데 사용
gpasswd	그룹에 구성원 및 관리자를 추가, 삭제하는 데 사용
groupadd	지정된 이름을 가진 그룹 생성
groupdel	지정된 이름을 가진 그룹 삭제
groupmems	유저가 슈퍼 유저 권한 없이 자신의 그룹 구성원 목록을 관리할 수 있도록 허용
groupmod	지정된 그룹의 이름 또는 GID를 수정하는 데 사용
grpck	그룹 파일 /etc/group과 /etc/gshadow의 무결성 확인
grpconv	일반 그룹 파일에서 Shadow 그룹 파일 생성, 업데이트
grpunconv	/etc/gshadow에서 /etc/group 업데이트 후 전자를 삭제
lastlog	모든 유저나 지정된 유저의 가장 최근 로그인 보고
login	시스템에서 사용자가 로그인할 수 있도록 함
logoutd	로그온 시간 및 포트에 대한 제한을 설정하는 데 쓰이는 데몬
newgidmap	유저 네임스페이스의 gid 매핑을 설정하는 데 사용
newgrp	로그인 세션 중에 현재 GID를 변경하는 데 사용
newuidmap	유저 네임스페이스의 uid 매핑을 설정하는 데 사용
newusers	일련의 전체 사용자 계정들을 만들거나 업데이트하는 데 사용
nologin	계정을 사용할 수 없다는 메시지를 표시함. 사용하지 않도록 설정된 계정의 기본 셸로 사용하도록 설계됨.
passwd	유저 또는 그룹 계정의 암호를 변경하는 데 사용
pwck	/etc/passwd와 /etc/shadow의 무결성 확인
pwconv	일반 암호 파일에서 Shadow 암호 파일 생성 또는 업데이트
pwunconv	/etc/shadow에서 /etc/passwd 업데이트 후 전자를 삭제

sg	유저의 GID가 지정된 그룹의 GID로 설정된 동안 지정된 명령 실행
su	다른 유저와 그룹 ID로 셸 실행
useradd	지정된 이름으로 새 유저 생성 또는 기본 새 유저 정보 업데이트
userdel	지정된 유저 계정 삭제
usermod	지정된 유저의 로그인 이름, UID, 셸, 초기 그룹, 홈 디렉토리 등을 수정하는 데 사용
vigr	/etc/group이나 /etc/gshadow 파일 수정
vipw	/etc/passwd나 /etc/shadow 파일 수정

6.25. GCC-9.2.0

GCC 패키지는 C, C++ 컴파일러가 들어있는 GNU 컴파일러 컬렉션을 포함하고 있다.

예상 빌드 시간: 88 SBU (테스트 포함)
필요 디스크 공간: 4.2 GB

6.25.1. GCC 설치

x86_64에서 빌드하고 있다면, 64비트 라이브러리의 기본 디렉토리 이름을 “lib”로 변경하라:

```
case $(uname -m) in
  x86_64)
    sed -e '/m64=/s/lib64/lib/' ₩
        -i.orig gcc/config/i386/t-linux64
  ;;
esac
```

Gcc 2단계에서처럼, Glibc-2.31에서 언급된 문제를 수정한다:

```
sed -e '1161 s|^|//|' ₩
    -i libsanitizer/sanitizer_common/sanitizer_platform_limits_posix.cc
```

GCC 문서는 전용 빌드 디렉토리에 GCC를 구축할 것을 권장한다:

```
mkdir -v build
cd       build
```

GCC 컴파일을 준비한다:

```
SED=sed                               ₩
../configure --prefix=/usr            ₩
             --enable-languages=c,c++ ₩
             --disable-multilib       ₩
             --disable-bootstrap      ₩
             --with-system-zlib
```

다른 언어의 경우는 아직 사용할 수 없는 몇 가지 전제 조건이 필요하다는 점을 참고하라. GCC에서 지원되는 모든 언어를 빌드하는 방법은 BLFS 책을 참고하라.

Configure 매개 변수들의 의미:

SED=sed
/tools/bin/sed로 경로가 하드코딩되는 것을 막는다.

--with-system-zlib
GCC가 자체적으로 갖는 내부 Zlib 라이브러리 대신 시스템에 설치된 Zlib 라이브러리에 링크되도록 설정한다.

패키지를 컴파일한다:

```
make
```

중요

이 절의 GCC 테스트 스위트는 매우 중요하다. 어떤 상황에서도 건너뛰지 않도록 하라.

GCC 테스트 스위트 중의 한 테스트 세트는 스택을 많이 차지하는 것으로 알려져 있으므로 테스트를 실행하기 전에 스택 크기를 늘린다:

```
ulimit -s 32768
```

권한 없는 유저로 결과를 테스트하되 오류 발생 시 중단하지 않도록 하라:

```
chown -Rv nobody .
su nobody -s /bin/bash -c "PATH=$PATH make -k check"
```

테스트 스위트 결과에 대한 요약을 확인하려면 다음을 실행하라:

```
../contrib/test_summary
```

요약만을 보려면, 위 결과를 grep -A7 Summ로 파이핑하라.

결과는 http://www.linuxfromscratch.org/lfs/build-logs/9.1/와 https://gcc.gnu.org/ml/gcc-testresults/에 있는 결과와 비교할 수 있다.

get_time과 관련된 6개의 테스트가 실패하는 것으로 알려져 있다. 이것들은 명백히 en_HK 로케일과 관련이 있다.

experimental/net에서 lookup.cc와 reverse.cc라는 두 테스트는 /etc/hosts와 iana-etc가 필요해서 LFS chroot 환경에서 실패하는 것으로 알려져 있다.

pr57193.c와 pr90178.c라는 두 가지 테스트가 실패하는 것으로 알려져 있다.

예기치 못한 실패가 더 있을 수도 있다. GCC 개발자들은 보통 이런 문제들을 알고 있지만 아직 해결하지 못했다. 위의 URL의 테스트 결과와 크게 다르지 않는 한, 계속 진행해도 무방하다.

패키지를 설치하고 필요하지않은 디렉토리를 삭제한다:

```
make install
rm -rf /usr/lib/gcc/$(gcc -dumpmachine)/9.2.0/include-fixed/bits/
```

GCC 빌드 디렉토리는 현재 nobody가 소유하며 설치된 헤더 디렉토리(와 그 내용)의 소유권은 불명확할 것이다. 소유권을 root 유저 및 그룹으로 변경하라:

```
chown -v -R root:root ₩
    /usr/lib/gcc/*linux-gnu/9.2.0/include{,-fixed}
```

"역사적인" 이유로 FHS에서 요구하는 심볼릭 링크를 생성한다.

```
ln -sv ../usr/bin/cpp /lib
```

많은 패키지들은 cc라는 이름으로 C 컴파일러를 호출한다. 이러한 패키지들에 대해 올바르게 동작할 수 있도록 심볼릭 링크를 생성하라:

```
ln -sv gcc /usr/bin/cc
```

링크 시간 최적화(Link Time Optimization, LTO)를 사용하여 프로그램을 빌드할 수 있도록 호환성 심볼릭 링크를 추가하라:

```
install -v -dm755 /usr/lib/bfd-plugins
ln -sfv ../../libexec/gcc/$(gcc -dumpmachine)/9.2.0/liblto_plugin.so ₩
        /usr/lib/bfd-plugins/
```

이제 우리의 최종 툴체인(toolchain)이 자리를 잡았으니 컴파일과 링크가 예상대로 잘 될 수 있도록 다시 한 번 확인하는 것이 중요하다. 따라서 앞 장에서 했던 것과 같은 온전성 검사를 수행한다:

```
echo 'int main(){}' > dummy.c
cc dummy.c -v -Wl,--verbose &> dummy.log
readelf -l a.out | grep ': /lib'
```

마지막 명령의 출력은(플랫폼에 따른 동적 링커 이름의 차이는 무관) 다음과 같으며 오류가 없어야 한다:

```
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
```

이제 올바른 시작 파일을 사용하도록 설정되었는지 확인하라:

```
grep -o '/usr/lib.*/crt[1in].*succeeded' dummy.log
```

이 명령의 출력은 다음과 같아야 한다:

```
/usr/lib/gcc/x86_64-pc-linux-gnu/9.2.0/../../../../lib/crt1.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/9.2.0/../../../../lib/crti.o succeeded
/usr/lib/gcc/x86_64-pc-linux-gnu/9.2.0/../../../../lib/crtn.o succeeded
```

당신의 시스템 아키텍쳐에 따라 위의 내용은 약간 다를 수 있는데, 대개 /usr/lib/gcc 다음에 나오는 디렉토리 이름이 차이난다. 여기서 중요하게 봐야 할 것은 gcc가 /usr/lib 디렉토리에서 crt*.o 파일 3개를 모두 찾아내었는지다.

컴파일러가 올바른 헤더 파일을 검색하는지 확인하라:

```
grep -B4 '^ /usr/include' dummy.log
```

이 명령은 다음과 같은 결과를 출력해야 한다:

```
#include <...> search starts here:
 /usr/lib/gcc/x86_64-pc-linux-gnu/9.2.0/include
 /usr/local/include
 /usr/lib/gcc/x86_64-pc-linux-gnu/9.2.0/include-fixed
 /usr/include
```

다시 말하지만, 당신의 대상 트리플렛 뒤의 디렉토리 이름은 아키텍쳐에 따라 위와 다를 수 있다.

그 다음, 새 링커가 올바른 검색 경로와 함께 사용되고 있는지 확인하라:

```
grep 'SEARCH.*/usr/lib' dummy.log |sed 's|; |₩n|g'
```

'-linux-gnu' 요소가 있는 경로에 대한 참조는 무시해도 되지만, 나머지 결과는 다음과 같아야 한다:

```
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib64")
SEARCH_DIR("/usr/local/lib64")
SEARCH_DIR("/lib64")
SEARCH_DIR("/usr/lib64")
SEARCH_DIR("/usr/x86_64-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

32비트 시스템에서는 디렉토리 몇 개가 다른 것을 볼 수 있다. 예를 들어, i686 시스템에서는 출력이 다음과 같다:

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib32")
SEARCH_DIR("/usr/local/lib32")
SEARCH_DIR("/lib32")
SEARCH_DIR("/usr/lib32")
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
SEARCH_DIR("/usr/lib");
```

다음으로 올바른 libc를 사용하고 있는지 확인하라:

```
grep "/lib.*/libc.so.6 " dummy.log
```

이 명령의 출력은 다음과 같아야 한다:

```
attempt to open /lib/libc.so.6 succeeded
```

마지막으로 GCC가 올바른 동적 링커를 사용하는지 확인하라:

```
grep found dummy.log
```

이 명령의 출력은 다음과 같아야 한다(플랫폼에 따른 동적 링커 이름의 차이는 무관):

```
found ld-linux-x86-64.so.2 at /lib/ld-linux-x86-64.so.2
```

만약 출력이 위와 같이 나타나지 않거나 출력이 전혀 없다면, 무언가 심각하게 잘못된 것이다. 이전 단계를 조사하고 추적해서 문제가 어디에 있는지 파악하고 수정하라. 가장 유력한 원인은 spec 파일 조정에 이상이 생겼기 때문이다. 어떤 문제든 계속 진행하기 전에 해결해야 할 것이다.

모든 것이 올바르게 작동하면 테스트 파일을 정리하라:

```
rm -v dummy.c a.out dummy.log
```

마지막으로, 위치가 잘못된 파일을 이동하라:

```
mkdir -pv /usr/share/gdb/auto-load/usr/lib
mv -v /usr/lib/*gdb.py /usr/share/gdb/auto-load/usr/lib
```

6.25.2. GCC의 내용

설치된 프로그램들:	c++, cc (gcc로 링크), cpp, g++, gcc, gcc-ar, gcc-nm, gcc-ranlib, gcov, gcov-dump, gcov-tool
설치된 라이브러리들:	libasan.{a,so}, libatomic.{a,so}, libcc1.so, libgcc.a, libgcc_eh.a, libgcc_s.so, libgcov.a, libgomp.{a,so}, libitm.{a,so}, liblsan.{a,so}, liblto_plugin.so, libquadmath.{a,so}, libssp.{a,so}, libssp_nonshared.a, libstdc++.{a,so}, libstdc++fs.a, libsupc++.a, libtsan.{a,so}, libubsan.{a,so}
설치된 디렉토리들:	/usr/include/c++, /usr/lib/gcc, /usr/libexec/gcc, /usr/share/gcc-9.2.0

요약

c++	C++ 컴파일러

cc	C 컴파일러
cpp	C 전처리기; 소스 파일에서 #include, #define 등의 유사한 구문들을 확장하기 위해 컴파일러에 의해 사용된다
g++	C++ 컴파일러
gcc	C 컴파일러
gcc-ar	명령줄에 플러그인을 추가하는 ar 래퍼(wrapper). 이 프로그램은 "링크 시간 최적화"를 추가하는 데만 사용되며 기본 빌드 옵션에서는 유용하지 않음
gcc-nm	명령줄에 플러그인을 추가하는 nm 래퍼. 이 프로그램은 "링크 시간 최적화"를 추가하는 데만 사용되며 기본 빌드 옵션에서는 유용하지 않음
gcc-ranlib	명령줄에 플러그인을 추가하는 ranlib 래퍼. 이 프로그램은 "링크 시간 최적화"를 추가하는 데만 사용되며 기본 빌드 옵션에서는 유용하지 않음
gcov	적용 범위 테스트 도구; 최적화의 영향이 가장 효과적인 지점을 파악하기 위해 프로그램을 분석하는데 사용됨
gcov-dump	오프라인 gcda 및 gcno 프로필 덤프 도구
gcov-tool	오프라인 gcda 프로필 처리 도구
libasan	Address Sanitizer 런타임 라이브러리
libatomic	GCC atomic 내장 런타임 라이브러리
libcc1	C 전처리 라이브러리
libgcc	gcc에 대한 런타임 라이브러리 포함
libgcov	이 라이브러리는 GCC의 프로파일링이 활성화될 때 프로그램에 링크된다.
libgomp	C/C++와 포트란에서 멀티 플랫폼 공유 메모리 병렬 프로그래밍을 위한 OpenMP API의 GNU 구현
liblsan	Leak Sanitizer 런타임 라이브러리
liblto_plugin	컴파일 단위 전체에 걸쳐 최적화를 수행할 수 있도록 하는 GCC의 링크 시간 최적화(Link Time Optimization, LTO) 플러그인
libquadmath	GCC Quad Precision 수학 라이브러리 API
libssp	GCC의 stack-smashing 보호 기능을 지원하는 루틴 포함
libstdc++	표준 C++ 라이브러리
libstdc++fs	ISO/IEC TS 18822:2015 파일 시스템 라이브러리
libsupc++	C++ 프로그래밍 언어 지원 루틴 제공
libtsan	The Thread Sanitizer 런타임 라이브러리
libubsan	The Undefined Behavior Sanitizer 런타임 라이브러리

6.26. Pkg-config-0.29.2

pkg-config 패키지에는 configure와 make 파일 실행 중 도구를 빌드하기 위한 include 경로 및/또는 라이브러리 경로를 전달하는 도구가 포함되어 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 30 MB

6.26.1. Pkg-config 설치

Pkg-config 컴파일을 준비한다:

```
./configure --prefix=/usr           ₩
            --with-internal-glib    ₩
            --disable-host-tool     ₩
            --docdir=/usr/share/doc/pkg-config-0.29.2
```

Configure 옵션들의 의미:

--with-internal-glib
LFS에서는 외부 버전을 사용할 수 없기 때문에 pkg-config가 Glib의 내부 버전을 사용하게 한다.

--disable-host-tool
pkg-config 프로그램에 대한 원치 않는 하드 링크를 생성하지 못하게 한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 다음을 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.26.2. Pkg-config의 내용

설치된 프로그램: pkg-config
설치된 디렉토리: /usr/share/doc/pkg-config-0.29.2

요약

pkg-config 지정된 라이브러리 또는 패키지에 대한 메타 정보 반환

6.27. Ncurses-6.2

Ncurses 패키지는 문자 화면의 터미널 독립적 처리를 위한 라이브러리를 포함하고 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 43 MB

6.27.1. Ncurses 설치

Configure에 의해 처리되지 않는 정적 라이브러리 설치를 비활성화 한다:

```
sed -i '/LIBTOOL_INSTALL/d' c++/Makefile.in
```

Ncurses 컴파일을 준비한다:

```
./configure --prefix=/usr           ₩
            --mandir=/usr/share/man ₩
            --with-shared           ₩
            --without-debug         ₩
            --without-normal        ₩
            --enable-pc-files       ₩
            --enable-widec
```

Configure 옵션들의 의미:

--enable-widec
일반 라이브러리(예: libncurses.so.6.2) 대신 와이드 문자 라이브러리(예: libncursesw.so.6.2)가 빌드된다. 이러한 와이드 문자 라이브러리는 멀티바이트 및 기존의 8비트 로케일에서 모두 사용할 수 있으며, 일반 라이브러리는 8비트 로케일에서만 제대로 작동한다. 와이드 문자 및 일반 라이브러리는 소스끼리는 호환되지만 바이너리 호환은 되지 않는다.

--enable-pc-files
pkg-config용 .pc 파일을 생성하고 설치한다.

--without-normal
대부분의 정적 라이브러리들의 빌드와 설치를 비활성화한다.

패키지를 컴파일한다:

```
make
```

이 패키지는 테스트 스위트가 있지만, 패키지가 설치된 후에만 실행할 수 있다. 테스트들은 test/ 디렉토리에 있다. 자세한 내용은 해당 디렉토리의 README 파일을 참고하라.

패키지를 설치한다:

```
make install
```

공유 라이브러리들을 원래 있어야 할 /lib 디렉토리로 이동한다:

```
mv -v /usr/lib/libncursesw.so.6* /lib
```

라이브러리가 이동되었기 때문에 존재하지 않는 파일을 가리키고 있는 심볼릭 링크 하나가 있다. 이것을 다시 만든다:

```
ln -sfv ../../lib/$(readlink /usr/lib/libncursesw.so) /usr/lib/libncursesw.so
```

많은 응용 프로그램들은 여전히 링커가 비-와이드 문자(즉, 일반 8비트 로케일용) Ncurses 라이브러리를 찾을 수 있는 것처럼 작동할 것이다. 이러한 응용 프로그램들이 와이드 문자 라이브러리에 링크되도록 심볼릭 링크 및 링커 스크립트로 트릭을 쓴다:

```
for lib in ncurses form panel menu ; do
   rm -vf                    /usr/lib/lib${lib}.so
   echo "INPUT(-l${lib}w)" > /usr/lib/lib${lib}.so
   ln -sfv ${lib}w.pc        /usr/lib/pkgconfig/${lib}.pc
done
```

마지막으로, 빌드 시 -lcurses를 찾는 오래된 응용 프로그램들을 빌드 가능하도록 만든다:

```
rm -vf                     /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" > /usr/lib/libcursesw.so
ln -sfv libncurses.so      /usr/lib/libcurses.so
```

필요하다면, Ncurses 문서를 설치하라:

```
mkdir -v       /usr/share/doc/ncurses-6.2
cp -v -R doc/* /usr/share/doc/ncurses-6.2
```

참고

소스로부터 컴파일하여 설치한 패키지는 런타임 중에 비-와이드 문자 Ncurses 라이브러리에 링크되지 않으므로, 위의 설명은 비-와이드 문자 Ncurses 라이브러리를 생성하지 않는다. 그러나 비-와이드 문자 Ncurses 라이브러리에 대해서만 링크하는 것으로 알려진 바이너리 전용 응용 프로그램들은 라이브러리 버전 5가 필요하다. 일부 바이너리 전용 응용 프로그램 때문에, 또는 LSB를 준수하기 위해 이러한 라이브러리가 필요하다면 다음 명령을 사용하여 패키지를 다시 빌드하라:

```
make distclean
./configure --prefix=/usr    ₩
            --with-shared    ₩
            --without-normal ₩
            --without-debug  ₩
            --without-cxx-binding ₩
            --with-abi-version=5
make sources libs
cp -av lib/lib*.so.5* /usr/lib
```

6.27.2. Ncurses의 내용

설치된 프로그램들:	captoinfo (tic로 링크), clear, infocmp, infotocap (tic로 링크), ncursesw6-config, reset (tset로 링크), tabs, tic, toe, tput, tset
설치된 라이브러리들:	libcursesw.so (libncursesw.so로 심볼릭 링크, 링커 스크립트), libformw.so, libmenuw.so, libncursesw.so, libncurses++w.a, libpanelw.so, 그리고 라이브러리 이름에 "w"가 없는 비-와이드 문자 대응 파일들.
설치된 디렉토리들:	/usr/share/tabset, /usr/share/terminfo, /usr/share/doc/ncurses-6.2

요약

captoinfo	termcap 내용을 terminfo 내용으로 변환

clear	화면을 지움(가능한 경우)
infocmp	terminfo 내용 비교 또는 출력
infotocap	terminfo 내용을 termcap 내용으로 변환
ncursesw6-config	ncurses에 대한 구성 정보 제공
reset	터미널을 기본값으로 다시 초기화
tabs	터미널의 탭 간격 초기화 및 설정
tic	terminfo entry-description 컴파일러는 terminfo 파일을 소스 형식에서 ncurses 라이브러리 루틴에 필요한 바이너리 형식으로 변환한다[terminfo 파일에는 특정 터미널의 기능에 대한 정보가 포함되어 있다.]
toe	사용 가능한 모든 터미널 유형을 나열하고 각 터미널에 대한 기본 이름과 설명을 제공
tput	셸에서 터미널 고유 기능들을 사용할 수 있게 한다; 터미널을 재설정 또는 초기화하거나 터미널 타입의 풀 네임을 보고하는 데도 사용할 수 있음
tset	터미널을 초기화하는 데 사용
libcursesw	libncursesw로 링크
libncursesw	터미널 화면에 여러 가지 복잡한 방법으로 텍스트를 표시하는 기능을 포함. 이러한 기능을 사용하는 좋은 예는 커널의 make menuconfig동안 표시되는 메뉴이다.
libformw	양식 구현 함수 포함
libmenuw	메뉴 구현 함수 포함
libpanelw	패널 구현 함수 포함

6.28. Libcap-2.31

Libcap 패키지는 리눅스 커널에서 사용할 수 있는 POSIX 1003.1e 기능에 대한 사용자 공간 인터페이스를 구현한다. 이러한 기능들은 강력한 모든 root 권한을 별개의 권한 집합들로 분할한다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 8.5 MB

6.28.1. Libcap 설치

두 정적 라이브러리들이 설치되지 않게 한다:

```
sed -i '/install.*STA...LIBNAME/d' libcap/Makefile
```

패키지를 컴파일한다:

```
make lib=lib
```

make 옵션의 의미:

lib=lib
x86_64에서 라이브러리 디렉토리를 /lib64가 아닌 /lib로 설정한다. x86에는 영향이 없다.

결과를 테스트하려면 실행하라:

```
make test
```

패키지를 설치한다:

```
make lib=lib install
chmod -v 755 /lib/libcap.so.2.31
```

6.28.2. Libcap의 내용

설치된 프로그램들: capsh, getcap, getpcaps, setcap
설치된 라이브러리들: libcap.so, libpsx.a

요약

capsh	기능 지원을 탐색하고 제한하는 셸 래퍼
getcap	파일 기능 검사
getpcaps	쿼리된 프로세스(들)의 기능 표시
setcap	파일 기능 설정
libcap	POSIX 1003.1e 기능을 조작하는 라이브러리 함수 포함
libpsx	pthread 라이브러리와 관련된 syscall에 대해 POSIX semantics를 지원하는 함수를 포함

6.29. Sed-4.8

Sed 패키지는 스트림 에디터를 포함하고 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 34 MB

6.29.1. Sed 설치

먼저 LFS 환경에서의 문제 하나를 해결하고 실패하는 테스트를 제거한다:

```
sed -i 's/usr/tools/'                 build-aux/help2man
sed -i 's/testsuite.panic-tests.sh//' Makefile.in
```

Sed 컴파일을 준비한다:

```
./configure --prefix=/usr --bindir=/bin
```

패키지를 컴파일 하고 HTML 문서를 생성한다:

```
make
make html
```

결과를 테스트하려면, 실행하라:

```
make check
```

패키지와 문서를 설치한다:

```
make install
install -d -m755           /usr/share/doc/sed-4.8
install -m644 doc/sed.html /usr/share/doc/sed-4.8
```

6.29.2. Sed의 내용

설치된 프로그램: sed
설치된 디렉토리: /usr/share/doc/sed-4.8

요약

sed 텍스트 파일을 한번에 필터링 및 변환

6.30. Psmisc-23.2

Psmisc 패키지는 실행 중인 프로세스에 대한 정보를 표시하는 프로그램을 포함하고 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 4.6 MB

6.30.1. Psmisc 설치

Psmisc 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

이 패키지는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

마지막으로, killall과 fuser 프로그램을 FHS에서 지정한 위치로 이동하라:

```
mv -v /usr/bin/fuser   /bin
mv -v /usr/bin/killall /bin
```

6.30.2. Psmisc의 내용

설치된 프로그램들: fuser, killall, peekfd, prtstat, pslog, pstree, pstree.x11 (pstree로 링크)

요약

fuser	지정된 파일 또는 파일 시스템을 사용하는 프로세스의 ID(PID) 보고
killall	이름으로 프로세스를 중단; 지정된 명령을 실행하는 모든 프로세스에 신호를 전송
peekfd	PID가 지정되면 실행 중인 프로세스의 파일 디스크립터를 훑어봄
prtstat	프로세스의 정보를 출력
pslog	프로세스의 현재 로그 경로 보고
pstree	실행 중인 프로세스를 트리로 표시
pstree.x11	종료하기 전에 확인 응답을 기다리는 것을 제외하면 pstree와 동일

6.31. Iana-Etc-2.30

Iana-Etc 패키지는 네트워크 서비스와 프로토콜에 대한 데이터를 제공한다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 2.3 MB

6.31.1. Iana-Etc 설치

다음 명령은 IANA가 제공하는 raw 데이터를 /etc/protocols와 /etc/services 데이터 파일에 대해 올바른 형식으로 변환한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

6.31.2. Iana-Etc의 내용

설치된 파일들: /etc/protocols와 /etc/services

요약

/etc/protocols	TCP/IP 하위 시스템에서 사용할 수 있는 다양한 DARPA 인터넷 프로토콜을 기술
/etc/services	인터넷 서비스를 위한 친숙한 텍스트 이름과 그 기저에 할당된 포트 번호 및 프로토콜 유형 간의 매핑 제공

6.32. Bison-3.5.2

Bison 패키지에는 파서 생성기가 포함되어 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 43 MB

6.32.1. Bison 설치

Bison 컴파일을 준비한다:

```
./configure --prefix=/usr --docdir=/usr/share/doc/bison-3.5.2
```

패키지를 컴파일한다:

```
make
```

bison과 flex 사이에는 순환 의존성이 있다. 원한다면, 다음 절에서 flex를 설치한 후, bison 패키지를 재빌드하고 make check를 통해 bison 검사를 수행할 수 있다.

패키지를 설치한다:

```
make install
```

6.32.2. Bison의 내용

설치된 프로그램들: bison과 yacc
설치된 라이브러리: liby.a
설치된 디렉토리: /usr/share/bison

요약

bison 일련의 규칙으로부터 텍스트 파일의 구조를 분석하는 프로그램 생성, Bison은 Yacc(Yet Another Compiler Compiler)의 대체 프로그램

yacc bison의 래퍼, bison 대신 yacc를 호출하는 프로그램; -y 옵션과 함께 bison을 호출

liby Yacc 호환 yyerror 및 main 함수를 포함하는 Yacc 라이브러리; 이 라이브러리는 일반적으로 그다지 유용하지는 않지만, POSIX에 필요하다

6.33. Flex-2.6.4

Flex 패키지에는 텍스트의 패턴을 인식하는 프로그램을 생성하는 도구가 포함되어 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 36 MB

6.33.1. Flex 설치

우선, Glibc-2.26에서 생긴 문제를 수정한다:

```
sed -i "/math.h/a #include <malloc.h>" src/flexdef.h
```

이 빌드 절차는 help2man 프로그램이 실행 파일의 --help 옵션으로부터 man 페이지를 생성할 수 있다고 가정한다. 그러나 이는 존재하지 않기 때문에, 우리는 환경 변수를 사용하여 이 과정을 생략한다. 이제 Flex 컴파일을 준비한다:

```
HELP2MAN=/tools/bin/true ₩
./configure --prefix=/usr --docdir=/usr/share/doc/flex-2.6.4
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면(약 0.5 SBU) 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

몇몇 프로그램들은 아직 flex를 인식하지 못해서 이전 프로그램인 lex를 사용하려 할 것이다. 이러한 프로그램들을 지원하기 위해 lex 에뮬레이션 모드에서 flex를 실행하는 lex라는 심볼릭 링크를 생성한다:

```
ln -sv flex /usr/bin/lex
```

6.33.2. Flex의 내용

설치된 프로그램들: flex, flex++ (flex로 링크), lex (flex로 링크)
설치된 라이브러리: libfl.so
설치된 디렉토리: /usr/share/doc/flex-2.6.4

요약

flex	텍스트 패턴 인식 프로그램 생성 도구; 융통성있게 패턴 검색 규칙을 지정하여, 전문 프로그램 개발의 필요성을 근절한다
flex++	C++ 코드 및 클래스를 생성하는 데 사용되는 flex의 확장. flex로의 심볼릭 링크이다
lex	lex 에뮬레이션 모드에서 flex를 실행하는 심볼릭 링크
libfl	flex 라이브러리

6.34. Grep-3.4

Grep 패키지는 파일 내용을 검색하는 프로그램을 포함하고 있다.

예상 빌드 시간: 0.7 SBU
필요 디스크 공간: 39 MB

6.34.1. Grep 설치

Grep 컴파일을 준비한다:

```
./configure --prefix=/usr --bindir=/bin
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.34.2. Grep의 내용

설치된 프로그램들: egrep, fgrep, grep

요약

egrep 확장 정규 표현식으로 검색된 라인 출력
fgrep 고정 문자열 목록으로 검색된 라인 출력
grep 일반 정규 표현식으로 검색된 라인 출력

6.35. Bash-5.0

Bash 패키지에는 Bourne-Again SHell이 포함되어 있다.

예상 빌드 시간: 1.9 SBU
필요 디스크 공간: 62 MB

6.35.1. Bash 설치

업스트림 수정 사항 몇 가지를 적용한다:

```
patch -Np1 -i ../bash-5.0-upstream_fixes-1.patch
```

Bash 컴파일을 준비한다:

```
./configure --prefix=/usr                    ₩
            --docdir=/usr/share/doc/bash-5.0 ₩
            --without-bash-malloc            ₩
            --with-installed-readline
```

Configure 옵션의 의미:

--with-installed-readline
Bash가 자체에 내장된 readline 대신 이미 시스템에 설치된 readline 라이브러리를 사용하도록 지시한다.

패키지를 컴파일한다:

```
make
```

테스트 스위트를 실행하지 않을 것이라면 “패키지 설치”로 이동하라.

테스트를 준비하려면, nobody 유저에게 소스 트리에 쓰기 권한을 부여하라:

```
chown -Rv nobody .
```

이제 nobody 유저로 테스트를 실행하라:

```
su nobody -s /bin/bash -c "PATH=$PATH HOME=/home make tests"
```

패키지를 설치하고 기본 실행 파일을 /bin으로 이동하라:

```
make install
mv -vf /usr/bin/bash /bin
```

새로 컴파일된 bash 프로그램을 실행한다(현재 실행 중인 프로그램 대체):

```
exec /bin/bash --login +h
```

참고

이 매개 변수는 bash 프로세스를 대화형 로그인 셸로 만들고 해싱을 비활성화한 채로 유지해서 새로운 프로그램을 사용할 수 있게 한다.

6.35.2. Bash의 내용

설치된 프로그램들: bash, bashbug, sh (bash로 링크)
설치된 디렉토리들: /usr/include/bash, /usr/lib/bash, /usr/share/doc/bash-5.0

요약

bash	널리 사용되는 명령어 인터프리터; 명령줄을 실행하기 전에 주어진 명령에 대해 다양한 종류의 확장 및 대체 작업을 수행하는 강력한 인터프리터
bashbug	사용자가 bash에 관련된 표준 형식의 버그 리포트를 작성하고 메일로 보낼 수 있도록 지원하는 셸 스크립트
sh	bash 프로그램에 대한 심볼릭 링크; sh로 호출되면, bash는 POSIX 표준을 준수하면서 가능한 한 과거 버전의 sh의 시작 동작을 모방한다

6.36. Libtool-2.4.6

Libtool 패키지에는 GNU 일반 라이브러리 지원 스크립트가 포함되어 있다. 공유 라이브러리를 사용하는 복잡성을 일관적이고 포팅 가능한 인터페이스로 래핑한다.

예상 빌드 시간: 1.8 SBU
필요 디스크 공간: 43 MB

6.36.1. Libtool 설치

Libtool 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

참고

libtool의 테스트 시간은 멀티 코어 시스템에서 크게 단축될 수 있다. 이렇게 하려면 위의 라인에 TESTSUITEFLAGS=-j<N>을 추가하라. 예를 들어 -j4를 사용하면 테스트 시간을 60% 이상 줄일 수 있다.

LFS 빌드 환경에서는 순환 의존성 때문에 5가지 테스트가 실패하는 것으로 알려졌으나 automake가 설치된 이후에 다시 확인하면 모든 테스트가 통과된다.

패키지를 설치한다:

```
make install
```

6.36.2. Libtool의 내용

설치된 프로그램들: libtool, libtoolize
설치된 라이브러리: libltdl.so
설치된 디렉토리들: /usr/include/libltdl, /usr/share/libtool

요약

libtool	일반화된 라이브러리 구축 지원 서비스 제공
libtoolize	패키지에 libtool 지원을 추가하는 표준적인 방법 제공
libltdl	dlopen 라이브러리의 다양한 애로 사항들을 감춤

6.37. GDBM-1.18.1

GDBM 패키지에는 GNU 데이터베이스 관리자가 포함되어 있다. 확장 가능한 해싱을 사용하고 표준 UNIX dbm과 유사하게 작동하는 데이터베이스 함수들의 라이브러리이다. 이 라이브러리는 키/데이터 쌍을 저장하고 키로 데이터를 검색하고 데이터와 함께 키를 삭제하기 위한 원시 요소들을 제공한다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 11 MB

6.37.1. GDBM 설치

GDBM 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --enable-libgdbm-compat
```

Configure 옵션의 의미:

--enable-libgdbm-compat
LFS 외부의 일부 패키지에는 예전 DBM 루틴이 필요하기 때문에 이 옵션으로 libgdbm 호환 라이브러리를 빌드한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.37.2. GDBM의 내용

설치된 프로그램들: gdbm_dump, gdbm_load, gdbmtool
설치된 라이브러리들: libgdbm.so와 libgdbm_compat.so

요약

gdbm_dump	GDBM 데이터베이스를 파일로 덤프
gdbm_load	덤프 파일에서 GDBM 데이터베이스 다시 생성
gdbmtool	GDBM 데이터베이스를 테스트하고 수정
libgdbm	해시 데이터베이스를 다루는 함수 포함
libgdbm_compat	이전 DBM 함수를 포함하는 호환성 라이브러리

6.38. Gperf-3.1

Gperf는 키 세트에서 완벽한 해시 함수를 생성한다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 6.3 MB

6.38.1. Gperf 설치

Gperf 컴파일을 준비한다:

```
./configure --prefix=/usr --docdir=/usr/share/doc/gperf-3.1
```

패키지를 컴파일한다:

```
make
```

여러 테스트를 동시에 실행하면(-j 옵션이 1 이상인) 테스트가 실패하는 것으로 알려져 있다. 결과를 테스트하려면 다음을 실행하라:

```
make -j1 check
```

패키지를 설치한다:

```
make install
```

6.38.2. Gperf의 내용

설치된 프로그램: gperf
설치된 디렉토리: /usr/share/doc/gperf-3.1

요약

gperf 키 세트로부터 완전한 해시를 생성

6.39. Expat-2.2.9

Expat 패키지는 XML 구문을 위한 스트림 지향 C 라이브러리를 포함하고 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 11 MB

6.39.1. Expat 설치

우선 LFS 환경에서의 회귀 테스트 문제를 해결한다:

```
sed -i 's|usr/bin/env |bin/|' run.sh.in
```

Expat 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --docdir=/usr/share/doc/expat-2.2.9
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

필요하다면 문서를 설치하라:

```
install -v -m644 doc/*.{html,png,css} /usr/share/doc/expat-2.2.9
```

6.39.2. Expat의 내용

설치된 프로그램: xmlwf
설치된 라이브러리: libexpat.so
설치된 디렉토리: /usr/share/doc/expat-2.2.9

요약

xmlwf	XML 문서가 제대로 구성되었는지의 여부를 확인할 수 있는 비검증 유틸리티
libexpat	XML을 파싱하기 위한 API 함수들을 포함

6.40. Inetutils-1.9.4

Inetutils 패키지는 기본적인 네트워킹을 위한 프로그램을 포함하고 있다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 29 MB

6.40.1. Inetutils 설치

Inetutils 컴파일을 준비한다:

```
./configure --prefix=/usr        ₩
            --localstatedir=/var ₩
            --disable-logger     ₩
            --disable-whois      ₩
            --disable-rcp        ₩
            --disable-rexec      ₩
            --disable-rlogin     ₩
            --disable-rsh        ₩
            --disable-servers
```

Configure 옵션들의 의미:

--disable-logger
시스템 로그 데몬에 메시지를 전달하는 스크립트가 사용하는 logger 프로그램을 설치하지 않도록 한다. Util-리눅스가 더 최신 버전을 설치하므로 이것을 설치하지 않도록 하라.

--disable-whois
오래돼서 쓰이지 않는 whois 클라이언트 빌드를 비활성화한다. 더 나은 whois 클라이언트 설명은 BLFS 책에 있다.

--disable-r*
보안 문제로 인해 사용되어서는 안 되는 구식 프로그램을 빌드하지 않도록 한다. 이런 프로그램들이 제공하는 기능들은 BLFS 책에 있는 openssh 패키지로 대체할 수 있다.

--disable-servers
Inetutils 패키지의 일부로 포함된 다양한 네트워크 서버의 설치가 비활성화된다. 이런 서버들은 기본 LFS 시스템에서 적합하지 않다고 여겨진다. 일부는 기본적으로 안전하지 않으며 신뢰할 수 있는 네트워크에서만 안전하다고 판단할 수 있다. 이런 많은 서버들을 위한 더 나은 대체품을 쓸 수 있다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

참고

테스트 중 하나인 libls.sh는, 초기 chroot 환경에서 실패할 수도 있지만 LFS 시스템이 완성된 후 테스트를 다시 실행하면 통과한다. 호스트 시스템에 ipv6 기능이 없는 경우 ping-localhost.sh 하나가 실패한다.

패키지를 설치한다:

```
make install
```

/usr에 액세스할 수 없는 경우 사용할 수 있도록 일부 프로그램을 이동한다:

```
mv -v /usr/bin/{hostname,ping,ping6,traceroute} /bin
mv -v /usr/bin/ifconfig /sbin
```

6.40.2. Inetutils의 내용

설치된 프로그램들: dnsdomainname, ftp, ifconfig, hostname, ping, ping6, talk, telnet, tftp, traceroute

요약

dnsdomainname	시스템의 DNS 도메인 네임 표시
ftp	파일 전송 프로토콜 프로그램
hostname	호스트 이름 보고 또는 설정
ifconfig	네트워크 인터페이스 관리
ping	응답 요청 패킷 전송 및 응답에 걸리는 시간 보고
ping6	IPv6 네트워크 버전 ping
talk	다른 사용자와 대화하는 데 사용됨
telnet	TELNET 프로토콜 인터페이스
tftp	약식 파일 전송 프로그램
traceroute	작업 중인 호스트에서 네트워크의 다른 호스트로 이동하는 패킷의 경로를 추적하여 도중의 모든 중간 경유지(게이트웨이)를 표시

6.41. Perl-5.30.1

Perl 패키지에는 Practical Extraction and Report Language가 포함되어 있다.

예상 빌드 시간: 9.2 SBU
필요 디스크 공간: 272 MB

6.41.1. Perl 설치

우선 Perl의 configuration 파일 및 선택적 테스트 스위트에서 참조할 기본 /etc/hosts 파일을 생성하라:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

이 버전의 Perl은 이제 Compress::Raw::Zlib 와 Compress::Raw::BZip2 모듈들을 빌드한다. 기본적으로 Perl은 소스의 내부 사본으로 빌드를 한다. 다음 명령을 실행해 Perl이 시스템에 설치된 라이브러리들을 사용하도록 하라:

```
export BUILD_ZLIB=False
export BUILD_BZIP2=0
```

Perl 설정을 완전히 수동으로 하려면, 다음 명령에서 “-des” 옵션을 지워서 패키지가 빌드되는 방식을 직접 선택하라. 그 대신에 Perl이 자동으로 감지하는 기본값을 사용하려면, 아래 명령을 그대로 실행하라:

```
sh Configure -des -Dprefix=/usr                ₩
             -Dvendorprefix=/usr           ₩
             -Dman1dir=/usr/share/man/man1 ₩
             -Dman3dir=/usr/share/man/man3 ₩
             -Dpager="/usr/bin/less -isR"  ₩
             -Duseshrplib                  ₩
             -Dusethreads
```

Configure 옵션들의 의미:

-Dvendorprefix=/usr
패키지들이 perl 모듈을 설치할 위치를 지정한다.

-Dpager="/usr/bin/less -isR"
more 대신 less를 사용하도록 지정한다.

-Dman1dir=/usr/share/man/man1 -Dman3dir=/usr/share/man/man3
Groff가 아직 설치되지 않았기 때문에, Configure 스크립트는 Perl에 대한 man 페이지를 설치하지 않는다. man 페이지를 설치하기 위해서는, 이 매개 변수를 전달함으로써 설치를 강제한다.

-Duseshrplib
일부 perl 모듈에 필요한 공유 libperl을 빌드한다.

-Dusethreads
perl이 스레드를 지원하도록 빌드한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면(대략 11 SBU), 실행하라:

```
make test
```

패키지를 설치하고 정리하라:

```
make install
unset BUILD_ZLIB BUILD_BZIP2
```

6.41.2. Perl의 내용

설치된 프로그램들:	corelist, cpan, enc2xs, encguess, h2ph, h2xs, instmodsh, json_pp, libnetcfg, perl, perl5.30.1 (perl로 하드링크), perlbug, perldoc, perlivp, perlthanks (perlbug로 하드링크), piconv, pl2pm, pod2html, pod2man, pod2text, pod2usage, podchecker, podselect, prove, ptar, ptardiff, ptargrep, shasum, splain, xsubpp, zipdetails
설치된 라이브러리들:	여기 모두 나열할 수 없는 수많은 항목
설치된 디렉토리:	/usr/lib/perl5

요약

corelist	Module::CoreList의 명령줄 프론트엔드
cpan	명령줄에서 Comprehensive Perl Archive Network (CPAN)와 상호 작용
enc2xs	유니코드 문자 매핑 또는 Tcl 인코딩 파일로부터 인코드 모듈에 대한 Perl 확장을 빌드
encguess	하나 또는 여러 파일의 인코딩 유형 추측
h2ph	.h C 헤더 파일을 .ph Perl 헤더 파일로 변환
h2xs	.h C 헤더 파일을 Perl 확장으로 변환
instmodsh	설치된 Perl 모듈을 검사하기 위한 셸 스크립트. 설치된 모듈로부터 tarball을 만들 수 있음
json_pp	특정 입력 및 출력 형식 간에 데이터 변환
libnetcfg	libnet Perl 모듈을 구성하는데 사용할 수 있음
perl	C, sed, awk, sh의 가장 좋은 특징들을 하나의 다목적 언어로 조합
perl5.30.1	perl의 하드 링크
perlbug	Perl이나 그 모듈에 관련된 버그 리포트를 생성하고 메일로 보내는 데 사용
perldoc	Perl 설치 트리나 Perl 스크립트에 적힌 pod 형식으로 문서를 표시
perlivp	Perl 설치 검증 절차; Perl과 해당 라이브러리가 올바르게 설치되었는지 확인하는 데 사용할 수 있음
perlthanks	Perl 개발자들에게 보내는 감사 메시지를 생성하는 데 사용됨
piconv	문자 인코딩 변환기의 Perl 버전 iconv
pl2pm	Perl4 .pl 파일을 Perl5 .pm 모듈로 변환하기 위한 간단한 도구
pod2html	pod 형식에서 HTML 형식으로 파일 변환
pod2man	pod 데이터를 formatted *roff 입력으로 변환
pod2text	pod 데이터를 formatted ASCII 텍스트로 변환
pod2usage	파일에 포함된 pod 문서의 사용 메시지 출력
podchecker	pod 형식 문서 파일의 문법 확인
podselect	포드 문서의 선택된 섹션 표시
prove	Test::Harness 모듈에 대해 테스트를 실행하기 위한 명령줄 도구

ptar	Perl로 쓰여진 tar같은 프로그램
ptardiff	추출된 아카이브를 추출되지 않은 아카이브와 비교하는 Perl 프로그램
ptargrep	tar 아카이브의 파일 내용에 패턴 일치를 적용하는 Perl 프로그램
shasum	SHA 체크섬을 출력 및 확인
splain	Perl에서 상세한 경고 진단을 강제 적용하기 위해 사용
xsubpp	Perl XS 코드를 C 코드로 변환
zipdetails	Zip 파일의 내부 구조에 대한 세부 정보 표시

6.42. XML::Parser-2.46

XML::Parser 모듈은 James Clark의 XML 파서인 Expat에 대한 Perl 인터페이스이다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 2.4 MB

6.42.1. XML::Parser 설치

XML::Parser 컴파일을 준비한다:

```
perl Makefile.PL
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make test
```

패키지를 설치한다:

```
make install
```

6.42.2. XML::Parser의 내용

설치된 모듈: Expat.so

요약

Expat Perl Expat 인터페이스를 제공

6.43. Intltool-0.51.0

Intltool은 소스 파일에서 번역 가능한 문자열을 추출하는 데 사용되는 다국어화 도구이다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 1.5 MB

6.43.1. Intltool 설치

먼저 perl-5.22 이후 버전에서 발생하는 경고를 수정한다:

```
sed -i 's:₩₩₩${:₩₩₩$₩₩{:' intltool-update.in
```

Intltool 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
install -v -Dm644 doc/I18N-HOWTO /usr/share/doc/intltool-0.51.0/I18N-HOWTO
```

6.43.2. Intltool의 내용

설치된 프로그램들: intltool-extract, intltool-merge, intltool-prepare, intltool-update, intltoolize
설치된 디렉토리들: /usr/share/doc/intltool-0.51.0, /usr/share/intltool

요약

intltoolize	intltool을 사용하도록 패키지 준비
intltool-extract	gettext로 읽을 수 있는 헤더 파일 생성
intltool-merge	번역된 문자열을 다양한 파일 형식으로 병합
intltool-prepare	pot 파일을 업데이트하고 번역 파일과 병합
intltool-update	po 템플릿 파일을 업데이트하고 번역본과 병합

6.44. Autoconf-2.69

Autoconf 패키지에는 소스 코드를 자동으로 구성할 수 있는 셸 스크립트를 생성하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하(테스트 포함 약 3.2 SBU)
필요 디스크 공간: 79 MB

6.44.1. Autoconf 설치

먼저 Perl 5.28에서 생성된 버그를 수정한다.

```
sed '361 s/{/₩₩{/' -i bin/autoscan.in
```

Autoconf 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

테스트 스위트는 현재 bash-5와 libtool-2.4.3에 의해 고장났다. 그래도 테스트를 하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.44.2. Autoconf의 내용

설치된 프로그램들: autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate, ifnames
설치된 디렉토리: /usr/share/autoconf

요약

autoconf	소프트웨어 소스 코드 패키지를 자동으로 구성하여 다양한 종류의 Unix류 시스템에 맞게 적용되는 셸 스크립트 생성; 소프트웨어 소스 코드 패키지가 생성하는 config 스크립트는 독립적이다—이를 실행할 때는 autoconf 프로그램이 필요하지 않음
autoheader	사용할 구성을 위해 C #define문 템플릿 파일을 만드는 도구
autom4te	M4 매크로 프로세서의 래퍼
autoreconf	autoconf, autoheader, aclocal, automake, gettextize, libtoolize를 올바른 순서로 자동 실행하여 autoconf와 automake 템플릿 파일이 수정될 때 시간을 절약한다
autoscan	소프트웨어 패키지의 configure.in 파일을 작성하는 데 도움이 됨. 디렉토리 트리에서 원본 파일을 검사하여 일반적인 이식성 문제를 파악하고 패키지의 예비 configure.in 파일 역할을 하는 configure.scan 파일을 생성한다
autoupdate	이전 이름으로 autoconf 매크로를 계속 호출하는 configure.in 파일을 수정해서 현재 매크로 이름을 사용하도록 한다
ifnames	소프트웨어 패키지의 configure.in 파일을 작성할 때 도움이 됨. 패키지가 C 전처리기 조건부에서 사용하는 식별자를 출력한다. [패키지가 이미 이식성을 갖도록 설정되어 있으면, 이 프로그램

을 통해 어떤 configure을 확인해야 하는지 파악할 수 있다. autoscan에 의해 생성된 configure.in 파일의 공백도 채울 수 있다.]

6.45. Automake-1.16.1

Automake 패키지에는 Autoconf와 함께 사용할 Makefiles를 생성하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하(테스트 포함 약 8.1 SBU)
필요 디스크 공간: 107 MB

6.45.1. Automake 설치

Automake 컴파일을 준비한다:

```
./configure --prefix=/usr --docdir=/usr/share/doc/automake-1.16.1
```

패키지를 컴파일한다:

```
make
```

각각의 테스트마다 내부 딜레이가 있기 때문에, 프로세서가 하나뿐인 시스템에서도 -j4 make 옵션을 사용해서 테스트 속도를 높일 수가 있다. 결과를 테스트하려면 실행하라:

```
make -j4 check
```

LFS 환경에서 한 가지 테스트가 실패하는 것으로 알려져 있다: subobj.sh.

패키지를 설치한다:

```
make install
```

6.45.2. Automake의 내용

설치된 프로그램들: aclocal, aclocal-1.16 (aclocal와 하드 링크), automake, automake-1.16 (automake와 하드 링크)
설치된 디렉토리들: /usr/share/aclocal-1.16, /usr/share/automake-1.16, /usr/share/doc/automake-1.16.1

요약

aclocal	configure.in 파일의 내용에 따라 aclocal.m4 파일 생성
aclocal-1.16	aclocal로의 하드 링크
automake	Makefile.am 파일로부터 Makefile.in 파일을 자동으로 생성하는 도구 [패키지에 대한 모든 Makefile.in 파일을 만들려면 최상위 디렉토리에서 이 프로그램을 실행하라. configure.in 파일을 스캔하면, Makefile.am 파일을 자동으로 찾아 해당 Makefile.in 파일을 생성한다.]
automake-1.16	automake로의 하드 링크

6.46. Kmod-26

Kmod 패키지에는 커널 모듈을 로드하기 위한 라이브러리 및 유틸리티가 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 13 MB

6.46.1. Kmod 설치

Kmod 컴파일을 준비한다:

```
./configure --prefix=/usr        ₩
            --bindir=/bin        ₩
            --sysconfdir=/etc    ₩
            --with-rootlibdir=/lib ₩
            --with-xz            ₩
            --with-zlib
```

Configure 옵션들의 의미:

--with-xz, --with-zlib
Kmod가 압축된 커널 모듈을 처리할 수 있게 한다.

--with-rootlibdir=/lib
서로 다른 라이브러리 관련 파일을 올바른 디렉토리에 배치하도록 한다.

패키지를 컴파일한다:

```
make
```

이 패키지는 LFS chroot 환경에서 실행할 수 있는 테스트 스위트가 없다. 최소한 git 프로그램이 필요하며 여러 테스트가 git 리포지토리 밖에선 실행되지 않는다.

패키지를 설치하고 Module-Init-Tools(이전에 리눅스 커널 모듈을 처리하던 패키지)와 호환되도록 심볼릭 링크 생성한다:

```
make install

for target in depmod insmod lsmod modinfo modprobe rmmod; do
  ln -sfv ../bin/kmod /sbin/$target
done

ln -sfv kmod /bin/lsmod
```

6.46.2. Kmod의 내용

설치된 프로그램들: depmod (kmod로 링크), insmod (kmod로 링크), kmod, lsmod (kmod로 링크), modinfo (kmod로 링크), modprobe (kmod로 링크), and rmmod (kmod로 링크)
설치된 라이브러리: libkmod.so

요약

depmod 기존의 모듈 세트에서 찾은 심볼들을 기반으로 의존성 파일 생성; 이 의존성 파일은 modprobe에서 필요한 모듈을 자동으로 로드하는 데 사용된다

insmod	실행 중인 커널에 로드 가능한 모듈을 설치
kmod	커널 모듈 로드 및 언로드
lsmod	현재 로드된 모듈 나열
modinfo	커널 모듈과 연결된 오브젝트 파일을 검사하여 수집할 수 있는 모든 정보 표시
modprobe	depmod에 의해 생성된 의존성 파일을 사용하여, 관련 모듈을 자동으로 로드
rmmod	실행 중인 커널에서 모듈 언로드
libkmod	다른 프로그램에서 커널 모듈을 로드 및 언로드하는 데 사용되는 라이브러리

6.47. Gettext-0.20.1

Gettext 패키지에는 다국어화와 현지화를 위한 유틸리티가 포함되어 있다. 이를 통해 프로그램을 현지 언어를 지원(Native Language Support, NLS)하도록 컴파일할 수 있어 사용자의 모국어로 메시지를 출력할 수 있다.

예상 빌드 시간: 2.7 SBU
필요 디스크 공간: 249 MB

6.47.1. Gettext 설치

Gettext 컴파일을 준비한다:

```
./configure --prefix=/usr    ₩
            --disable-static ₩
            --docdir=/usr/share/doc/gettext-0.20.1
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면(이 작업은 시간이 오래 걸린다, 약 3 SBUs), 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
chmod -v 0755 /usr/lib/preloadable_libintl.so
```

6.47.2. Gettext의 내용

설치된 프로그램들: autopoint, envsubst, gettext, gettext.sh, gettextize, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, recode-sr-latin, xgettext
설치된 라이브러리들: libasprintf.so, libgettextlib.so, libgettextpo.so, libgettextsrc.so, libtextstyle.so, preloadable_libintl.so
설치된 디렉토리들: /usr/lib/gettext, /usr/share/doc/gettext-0.20.1, /usr/share/gettext, /usr/share/gettext-0.19.8

요약

autopoint	표준 Gettext 인프라 파일을 원본 패키지로 복사
envsubst	셸 서식 문자열에서 환경 변수 대체
gettext	메시지 카탈로그에서 번역 자료를 찾아 원본 언어의 메시지를 사용자의 언어로 번역
gettext.sh	주로 gettext의 셸 함수 라이브러리 역할
gettextize	다국어화를 시작하기 위해 패키지의 지정된 최상위 디렉토리에 모든 표준 Gettext 파일들을 복사
msgattrib	속성에 따라 번역 카탈로그의 메시지들을 필터링하고 속성을 조작
msgcat	지정된 .po 파일 연결 및 병합

msgcmp	두 .po 파일을 비교하여 두 파일이 모두 동일한 msgid 문자열을 포함하고 있는지 확인
msgcomm	지정된 .po 파일들에서 공통적인 메시지를 찾음
msgconv	번역 카탈로그를 다른 문자 인코딩으로 변환
msgen	영어 번역 카탈로그 생성
msgexec	번역 카탈로그의 모든 번역에 명령을 적용
msgfilter	번역 카탈로그의 모든 번역에 필터를 적용
msgfmt	번역 카탈로그로부터 바이너리 메시지 카탈로그 생성
msggrep	지정된 패턴과 일치하거나 지정된 원본 파일에 속하는 번역 카탈로그의 모든 메시지 추출
msginit	사용자 환경의 값으로 메타데이터 정보를 초기화하는 새 .po 파일 생성
msgmerge	두 원시 번역문을 하나의 파일로 결합
msgunfmt	바이너리 메시지 카탈로그를 원시 번역 텍스트로 디컴파일
msguniq	번역 카탈로그에서 중복 번역 통합
ngettext	문법 형식이 숫자에 따라 달라지는 텍스트 메시지의 기본 언어 번역 표시
recode-sr-latin	세르비안 텍스트를 키릴 문자에서 라틴 문자로 리코딩
xgettext	첫 번째 번역 템플릿을 만들기 위해 지정된 원본 파일로부터 번역 가능한 메시지 라인을 추출
libasprintf	<string> 문자열 및 <iostream> 스트림과 함께 사용하기 위해 C 형식 출력 루틴을 C++ 프로그램에서 사용할 수 있도록 하는 autosprintf 클래스를 정의한다.
libgettextlib	다양한 Gettext 프로그램들이 사용하는 공통 루틴을 포함하는 private 라이브러리; 이것들은 일반적인 용도로 사용되지 않는다
libgettextpo	.po 파일을 처리하는 특수 프로그램 작성에 사용됨; 이 라이브러리는 Gettext와 함께 제공된 표준 응용 프로그램들(msgcomm, msgcmp, msgattrib, msgen)만으로는 충분하지 않을 때 사용됨
libgettextsrc	다양한 Gettext 프로그램에서 사용하는 일반적인 루틴을 포함하는 private 라이브러리; 이것들은 일반적인 용도로 사용되지 않음
libtextstyle	텍스트 스타일링 라이브러리
preloadable_libintl	번역되지 않은 메시지를 기록할 때 libintl을 보조하는 LD_PRELOAD에서 사용할 라이브러리

6.48. Elfutils-0.178의 Libelf

Libelf는 실행 가능하고 링크 가능한 포맷(Executable and Linkable Format, ELF) 파일들을 처리하기 위한 라이브러리이다.

예상 빌드 시간: 0.9 SBU
필요 디스크 공간: 124 MB

6.48.1. Libelf 설치

Libelf는 elfutils-0.178 패키지의 일부이다. 소스 tarball로 elfutils-0.178를 사용하라.

Libelf 컴파일을 준비한다:

```
./configure --prefix=/usr --disable-debuginfod
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

run-elfclassify.sh라는 테스트는 실패한다고 알려져 있다.

Libelf만 설치한다:

```
make -C libelf install
install -vm644 config/libelf.pc /usr/lib/pkgconfig
rm /usr/lib/libelf.a
```

6.48.2. Libelf의 내용

설치된 라이브러리: libelf.so
설치된 디렉토리: /usr/include/elfutils

6.49. Libffi-3.3

Libffi 라이브러리는 다양한 호출 규약에 대한 이식 가능한 고급 프로그래밍 인터페이스를 제공한다. 프로그래머가 런타임 중에 호출 인터페이스 설명에 적힌 어떤 함수든 호출할 수 있다.

예상 빌드 시간: 1.9 SBU
필요 디스크 공간: 10 MB

6.49.1. Libffi 설치

참고

GMP와 유사하게, libffi는 사용 중인 프로세서에 최적화 되어 빌드된다. 다른 시스템 용으로 빌드하는 경우 CFLAGS와 CXXFLAGS를 export해서 아키텍쳐를 위한 범용 빌드로 설정하라. 그러지 않으면 libffi에 연결된 모든 응용 프로그램에서 잘못된 작업 오류를 일으킨다.

libffi 컴파일을 준비한다:

```
./configure --prefix=/usr --disable-static --with-gcc-arch=native
```

Configure 옵션의 의미:

--with-gcc-arch=native
현재 시스템에 대해 gcc 최적화를 한다. 이것을 지정하지 않으면 대상 시스템을 추측해서 빌드되고 따라서 생성된 코드가 시스템에 대해 알맞지 않을 수가 있다. 생성된 코드를 네이티브 시스템에서 성능이 낮은 시스템으로 복사하려면, 성능이 낮은 시스템을 매개 변수로 사용하라. 대체 시스템 유형에 대한 자세한 내용은 gcc 매뉴얼의 x86 옵션들을 참고하라.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

test-callback.c에 관련된 6개의 테스트가 실패한다고 알려져 있다.

패키지를 설치한다:

```
make install
```

6.49.2. Libffi의 내용

설치된 라이브러리: libffi.so

요약

libffi libffi API 함수들을 포함하고 있다.

6.50. OpenSSL-1.1.1d

OpenSSL 패키지에는 암호화와 관련된 관리 도구와 라이브러리가 포함되어 있다. 이는 OpenSSH, 이메일 응용 프로그램 및 웹 브라우저(HTTPS 사이트 접근용)와 같은 다른 패키지에 암호화 기능을 제공하는 데 쓰인다.

예상 빌드 시간: 2.1 SBU
필요 디스크 공간: 146 MB

6.50.1. OpenSSL 설치

OpenSSL 컴파일을 준비한다:

```
./config --prefix=/usr        ₩
         --openssldir=/etc/ssl ₩
         --libdir=lib         ₩
         shared               ₩
         zlib-dynamic
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make test
```

20-test_enc.t의 테스트 한 개가 실패하는 것으로 알려져 있다.

패키지를 설치한다:

```
sed -i '/INSTALL_LIBS/s/libcrypto.a libssl.a//' Makefile
make MANSUFFIX=ssl install
```

필요하다면 문서를 설치하라:

```
mv -v /usr/share/doc/openssl /usr/share/doc/openssl-1.1.1d
cp -vfr doc/* /usr/share/doc/openssl-1.1.1d
```

6.50.2. OpenSSL의 내용

설치된 프로그램들: c_rehash와 openssl
설치된 라이브러리들: libcrypto.{so,a}와 libssl.{so,a}
설치된 디렉토리들: /etc/ssl, /usr/include/openssl, /usr/lib/engines, /usr/share/doc/openssl-1.1.1d

요약

c_rehash 디렉토리의 모든 파일을 검사하고 그 해시 값들에 심볼릭 링크를 추가하는 Perl 스크립트.

openssl 셸에서 OpenSSL의 암호화 라이브러리의 다양한 암호화 기능을 사용하기 위한 명령줄 도구이다. man 1 openssl에 설명되어 있는 다양한 기능들을 사용할 수 있다.

libcrypto.so 다양한 인터넷 표준에 쓰이는 광범위한 암호 알고리즘을 구현한다. 이 라이브러리에서 제공하는 서비스들은 SSL, TLS, S/MIME의 OpenSSL 구현에 쓰이며 OpenSSH, OpenPGP 및 기타 암호 표준을 구현하는 데도 사용되었다.

libssl.so	Transport Layer Security (TLS v1) 프로토콜 구현. man 3 ssl를 실행해서 볼 수 있는 풍부한 API와 문서를 제공한다.

6.51. Python-3.8.1

Python 3 패키지는 Python 개발 환경을 포함하고 있다. 객체 지향 프로그래밍, 스크립트 작성, 대형 프로그램 프로토타입 제작 또는 응용 프로그램 자체를 개발하는 데 유용하다.

예상 빌드 시간: 1.2 SBU
필요 디스크 공간: 426 MB

6.51.1. Python 3 설치

Python 컴파일을 준비한다:

```
./configure --prefix=/usr       ₩
            --enable-shared     ₩
            --with-system-expat ₩
            --with-system-ffi   ₩
            --with-ensurepip=yes
```

Configure 옵션들의 의미:

--with-system-expat
Expat의 시스템 버전과의 링크를 활성화 한다.

--with-system-ffi
libffi의 시스템 버전과의 링크를 활성화 한다.

--with-ensurepip=yes
pip 및 setuptools 패키징 프로그램을 빌드한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 make test를 실행하라. 네트워크 연결 또는 추가 패키지가 필요한 일부 테스트는 생략한다. 네트워크 구성이 아직 완료되지 않았기 때문에 test_normalization이라는 이름의 테스트가 실패한다. 전체적으로 보다 확실한 결과를 얻으려면 Python 3이 BLFS에서 재설치될 때 테스트를 다시 실행하면 된다.

패키지를 설치한다:

```
make install
chmod -v 755 /usr/lib/libpython3.8.so
chmod -v 755 /usr/lib/libpython3.so
ln -sfv pip3.8 /usr/bin/pip3
```

설치 명령들의 의미:

chmod -v 755 /usr/lib/libpython3.{8.,}so
라이브러리가 다른 라이브러리와 같은 권한을 갖도록 수정한다.

원한다면 사전 포맷된 설명서를 설치하라:

```
install -v -dm755 /usr/share/doc/python-3.8.1/html

tar --strip-components=1  ₩
    --no-same-owner       ₩
    --no-same-permissions ₩
    -C /usr/share/doc/python-3.8.1/html ₩
    -xvf ../python-3.8.1-docs-html.tar.bz2
```

문서 설치 명령들의 의미:

--no-same-owner and --no-same-permissions
설치될 파일들의 소유권과 사용 권한을 올바르게 지정한다. 이러한 옵션이 없으면 tar를 사용해서 패키지를 설치할 때 소유권과 권한으로 원제작자의 이름이 부여된다.

6.51.2. Python 3의 내용

설치된 프로그램들:	2to3, idle3, pip3, pydoc3, python3, python3-config
설치된 라이브러리:	libpython3.8.so, libpython3.so
설치된 디렉토리들:	/usr/include/python3.8, /usr/lib/python3, /usr/share/doc/python-3.8.1

요약

2to3	Python 2.x 소스 코드를 읽고 일련의 수정을 거쳐서 Python 3.x 코드로 변환하는 Python 프로그램이다.
idle3	Python 인식 GUI 편집기를 여는 래퍼 스크립트. 이 스크립트를 실행하려면 반드시 Python보다 먼저 Tk를 설치해서 Tkinter Python 모듈이 구축되도록 해야 한다.
pip3	Python용 패키지 설치 프로그램. pip를 사용하여 Python Package Index 및 기타 인덱스로부터 패키지를 설치할 수 있다.
pydoc3	Python 문서 도구.
python3	대화형, 객체 지향 인터프리터 프로그래밍 언어.

6.52. Ninja-1.10.0

Ninja는 속도에 초점을 맞춘 작은 빌드 시스템이다.

예상 빌드 시간: 0.3 SBU
필요 디스크 공간: 89 MB

6.52.1. Ninja 설치

ninja는 실행될 때 보통 최대한의 프로세스를 병렬로 실행한다. 기본값은 시스템의 코어 수 +2이다. 경우에 따라 CPU를 과열시키거나 시스템 메모리가 부족해질 수 있다. 명령줄에서 실행할 경우 -jN 매개 변수를 전달하면 병렬 프로세스 수가 제한되지만 일부 패키지는 ninja를 내부적으로 실행해서 -j 매개 변수를 전달하지 않는다.

아래의 선택적 절차를 따르면 사용자가 환경 변수인 NINJAJOBS를 통해 병렬 프로세스 수를 제한할 수 있다. 예를 들면:

```
export NINJAJOBS=4
```

명령은 병렬 프로세스를 4개로 제한한다.

원한다면 다음을 실행하여 환경 변수 NINJAJOBS를 사용하는 기능을 추가하라:

```
sed -i '/int Guess/a ₩
  int   j = 0;₩
  char* jobs = getenv( "NINJAJOBS" );₩
  if ( jobs != NULL ) j = atoi( jobs );₩
  if ( j > 0 ) return j;₩
' src/ninja.cc
```

다음 명령으로 Ninja를 빌드한다:

```
python3 configure.py --bootstrap
```

빌드 옵션의 의미:

--bootstrap
ninja가 현재 시스템에 대해 재빌드되도록 강제한다.

결과를 테스트하려면 실행하라:

```
./ninja ninja_test
./ninja_test --gtest_filter=-SubprocessTest.SetWithLots
```

패키지를 설치한다:

```
install -vm755 ninja /usr/bin/
install -vDm644 misc/bash-completion /usr/share/bash-completion/completions/ninja
install -vDm644 misc/zsh-completion  /usr/share/zsh/site-functions/_ninja
```

6.52.2. Ninja의 내용

설치된 프로그램들: ninja

요약

ninja Ninja 빌드 시스템.

6.53. Meson-0.53.1

Meson은 오픈 소스 빌드 시스템이다. 극단적으로 빠르고, 더욱 중요한 것은 매우 사용자 친화적이다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 31 MB

6.53.1. Meson 설치

다음 명령으로 Meson을 컴파일하라:

```
python3 setup.py build
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
python3 setup.py install --root=dest
cp -rv dest/* /
```

설치 매개 변수의 의미:

--root=dest
기본적으로 python3 setup.py install은 다양한 파일(예: man 페이지)들을 Python Eggs에 설치한다. 최상위 위치가 지정되면 setup.py는 이러한 파일들을 표준 계층 구조로 설치한다. 그러면 그 계층을 그대로 복사해서 파일들이 표준 위치에 있도록 할 수 있다.

6.53.2. Meson의 내용

설치된 프로그램들: meson
설치된 디렉토리: /usr/lib/python3.8/site-packages/meson-0.53.1-py3.8.egg-info와 /usr/lib/python3.8/site-packages/mesonbuild

요약

meson 생산성이 높은 빌드 시스템

6.54. Coreutils-8.31

Coreutils 패키지는 기본적인 시스템 특성을 보여주고 설정하기 위한 유틸리티를 포함하고 있다.

예상 빌드 시간: 2.3 SBU
필요 디스크 공간: 202 MB

6.54.1. Coreutils 설치

POSIX를 사용하려면 Coreutils의 프로그램들이 멀티바이트 문자에서도 문자 경계를 정확하게 인식해야 한다. 다음 패치는 이 미준수 사항과 기타 다국어화 관련 버그를 수정한다.

```
patch -Np1 -i ../coreutils-8.31-i18n-1.patch
```

참고

과거에는 이 패치에서 많은 버그가 발견되었다. Coreutils 관리자에게 새 버그를 보고하려면 이 패치 없이도 버그가 재현 가능한지 먼저 확인해주기 바란다.

일부 시스템에서 무한 루프에 빠질 수 있는 한 테스트를 숨긴다:

```
sed -i '/test.lock/s/^/#/' gnulib-tests/gnulib.mk
```

이제 Coreutils 컴파일을 준비한다:

```
autoreconf -fiv
FORCE_UNSAFE_CONFIGURE=1 ./configure ₩
            --prefix=/usr            ₩
            --enable-no-install-program=kill,uptime
```

Configure 옵션들의 의미:

autoreconf
생성된 설정 파일들을 최신 버전의 automake와 일치하도록 업데이트한다.

FORCE_UNSAFE_CONFIGURE=1
패키지를 root 유저로 빌드할 수 있도록 한다.

--enable-no-install-program=kill,uptime
나중에 다른 패키지가 설치할 바이너리를 Coreutils가 설치하지 않도록 한다.

패키지를 컴파일한다:

```
make
```

테스트 스위트를 실행하지 않으려면 “패키지 설치”로 이동하라.

이제 테스트 스위트를 실행할 준비가 되었다. 먼저 root 유저로 실행해야 하는 테스트를 실행하라:

```
make NON_ROOT_USERNAME=nobody check-root
```

나머지 테스트는 nobody 유저로 실행할 것이다. 그러나 어떤 테스트들은 유저가 둘 이상의 그룹 구성원이어야 한다. 이런 테스트들을 건너뛰지 않도록 임시 그룹을 추가해서 nobody 유저가 해당 그룹에 속하도록 만든다:

```
echo "dummy:x:1000:nobody" >> /etc/group
```

root 사용자가 아닌 유저도 다음 테스트를 컴파일하고 실행할 수 있도록 일부 권한을 수정한다:

```
chown -Rv nobody .
```

이제 검사를 실행한다. su 환경의 PATH에 /tools/bin이 포함되어 있는지 확인하라.

```
su nobody -s /bin/bash ₩
        -c "PATH=$PATH make RUN_EXPENSIVE_TESTS=yes check"
```

테스트 프로그램인 test-getlogin은 chroot 환경처럼 부분적으로 빌드된 시스템 환경에서 실패한다고 알려져 있지만, 이 장을 마치고 실행하면 통과될 것이다. 테스트 프로그램 tty.sh도 실패한 것으로 알려져 있다.

임시 그룹을 삭제하라:

```
sed -i '/dummy/d' /etc/group
```

패키지를 설치한다:

```
make install
```

프로그램을 FHS에서 지정한 위치로 이동한다:

```
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,ln,ls,mkdir,mknod,mv,pwd,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
mv -v /usr/share/man/man1/chroot.1 /usr/share/man/man8/chroot.8
sed -i s/₩"1₩"/₩"8₩"/1 /usr/share/man/man8/chroot.8
```

LFS-부트스크립트 패키지의 스크립트 중 몇 가지는 head, nice, sleep, touch가 필요하다. 부팅 중에는 /usr에 접근할 수 없으므로 FHS를 준수하기 위해서는 이 바이너리들이 root 파티션에 있어야 한다:

```
mv -v /usr/bin/{head,nice,sleep,touch} /bin
```

6.54.2. Coreutils의 내용

설치된 프로그램들:	[, b2sum, base32, base64, basename, basenc, cat, chcon, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, id, install, join, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mktemp, mv, nice, nl, nohup, nproc, numfmt, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, realpath, rm, rmdir, runcon, seq, sha1sum, sha224sum, sha256sum, sha384sum, sha512sum, shred, shuf, sleep, sort, split, stat, stdbuf, stty, sum, sync, tac, tail, tee, test, timeout, touch, tr, true, truncate, tsort, tty, uname, unexpand, uniq, unlink, users, vdir, wc, who, whoami, yes
설치된 라이브러리:	(/usr/libexec/coreutils의) libstdbuf.so
설치된 디렉토리:	/usr/libexec/coreutils

요약

base32	base32 사양에 따라 데이터 인코딩 및 디코딩(RFC 4648)

base64	base64 사양에 따라 데이터 인코딩 및 디코딩(RFC 4648)
b2sum	BLAKE2 (512-bit) 체크섬을 출력하거나 확인
basename	파일 이름에서 경로 및 지정된 접미사 제거
basenc	다양한 알고리즘을 사용하여 데이터 인코딩 또는 디코딩
cat	파일을 표준 출력에 연결
chcon	파일 및 디렉토리에 대한 보안 설정 변경
chgrp	파일 및 디렉토리의 그룹 소유권 변경
chmod	각 파일의 사용 권한을 지정된 모드로 변경; 이 모드는 변경 사항을 뜻하는 문자나 새 사용 권한을 설명하는 8진수로 나타낼 수 있다.
chown	파일 및 디렉토리의 유저 및/또는 그룹 소유권 변경
chroot	지정된 디렉토리를 / 디렉토리인 것처럼 인식하게 해서 명령 실행
cksum	순환 중복 검사(Cyclic Redundancy Check, CRC) 체크섬과 지정된 각 파일의 바이트 수 출력
comm	두 개의 정렬된 파일을 비교하여 고유한 행과 공통적으로 갖는 행을 세 열로 출력
cp	파일들을 복사
csplit	지정된 파일을 여러 개의 새 파일로 분할하여 지정된 패턴 또는 행 번호에 따라 분리하고 각 새 파일의 바이트 수를 출력
cut	주어진 필드나 위치에 따라 행의 일부를 골라서 출력
date	현재 시간을 지정된 형식으로 표시하거나 시스템 날짜를 설정
dd	지정된 블록 크기와 갯수를 사용해서 파일을 복사. 동시에 데이터를 변환하는 것도 가능하다.
df	마운트된 모든 파일 시스템이나, 지정한 파일이 포함된 파일 시스템의 사용 가능한 디스크 공간을 출력
dir	지정된 각 디렉토리의 내용을 나열(ls 명령과 동일)
dircolors	LS_COLOR 환경 변수를 설정해서 ls에서 사용하는 색 설정을 변경하는 명령 출력
dirname	파일 이름에서 디렉토리 이외의 접미사 제거
du	현재 디렉토리, 지정된 디렉토리(모든 하위 디렉토리 포함) 또는 지정된 파일 각각에 사용되는 디스크 공간 출력
echo	지정된 문자열 표시
env	변경된 환경에서 명령 실행
expand	탭 문자를 공백 문자로 변환
expr	정규 표현식의 값을 표준 출력으로 출력
factor	지정된 모든 정수의 소인수를 출력
false	아무 작업도 수행하지 않음; 항상 실패를 나타내는 상태 코드와 함께 종료됨
fmt	지정된 파일의 단락 형식을 수정
fold	지정된 파일에서 줄 바꿈을 수행
groups	유저의 그룹 구성원 자격 출력
head	지정된 각 파일의 처음 열 줄(또는 지정된 줄 수) 출력
hostid	호스트의 숫자 식별자(16진수) 출력

id	현재 사용자나 지정된 사용자의 유효한 유저 ID, 그룹 ID, 그룹 구성원 자격 출력
install	권한 모드와 가능하면 소유자 및 그룹을 설정하면서 파일을 복사
join	두 개의 개별 파일에서 동일한 결합 필드를 가진 행끼리 합침
link	파일에 지정된 이름으로 하드 링크 생성
ln	파일 간의 하드 링크 또는 소프트(심볼릭) 링크 생성
logname	현재 사용자의 로그인 이름 출력
ls	지정된 각 디렉토리의 내용 나열
md5sum	Message Digest 5 (MD5) 체크섬 출력 또는 확인
mkdir	지정된 이름으로 디렉토리 생성
mkfifo	지정된 이름을 사용하여 UNIX 용어로 "named pipe"인 FIFO(First-In, First-Outs) 생성
mknod	지정된 이름으로 장치 노드 생성; 장치 노드는 문자 특수 파일, 블록 특수 파일 또는 FIFO이다.
mktemp	안전한 방식으로 임시 파일 생성; 스크립트에 사용됨
mv	파일 또는 디렉토리 이동 또는 이름 변경
nice	수정된 예약 우선 순위로 프로그램 실행
nl	지정된 파일로부터 줄 번호를 매김
nohup	프로세스가 중단되지 않도록 명령을 백그라운드에서 실행하고 출력을 로그 파일로 저장
nproc	사용할 수 있는 프로세싱 유닛의 갯수 출력
numfmt	숫자를 사람이 읽기 쉬운 문자열로 또는 문자열을 숫자로 변환
od	8진수 및 기타 형식의 파일 덤프
paste	지정된 파일들을 병합하여 각각의 행들을 탭 문자로 구분해 나란히 결합
pathchk	파일 이름이 유효한지 또는 이식 가능한지 확인
pinky	가벼운 finger 클라이언트; 지정된 유저에 대한 몇 가지 정보들을 출력
pr	인쇄할 파일 페이지와 열 형식 지정
printenv	환경 변수들을 출력
printf	C printf 함수처럼 지정된 형식에 따라 지정된 인수 출력
ptx	지정된 파일의 내용에서 각 키워드를 포함하는 순열 인덱스 생성
pwd	현재 작업 디렉토리의 이름 출력
readlink	지정된 심볼릭 링크의 값을 출력
realpath	실제 경로 출력
rm	파일이나 디렉토리 삭제
rmdir	디렉토리가 비어있으면 삭제
runcon	지정된 보안 설정을 사용해서 명령 실행
seq	지정된 범위 내에서 지정된 공차를 갖는 수열 출력
sha1sum	160비트 Secure Hash Algorithm 1 (SHA1) 체크섬 출력 또는 확인
sha224sum	224비트 Secure Hash Algorithm 1 체크섬 출력 또는 확인
sha256sum	256비트 Secure Hash Algorithm 1 체크섬 출력 또는 확인

sha384sum	384비트 Secure Hash Algorithm 1 체크섬 출력 또는 확인
sha512sum	512비트 Secure Hash Algorithm 1 체크섬 출력 또는 확인
shred	지정된 파일을 복잡한 패턴으로 반복적으로 덮어써서 데이터 복구가 어렵게 함
shuf	텍스트 행 순서를 섞음
sleep	지정된 시간 동안 일시 중지
sort	지정된 파일에서 행 정렬
split	지정된 파일을 크기 또는 줄 수에 따라 여러 개로 분할
stat	파일이나 파일 시스템의 상태 표시
stdbuf	표준 스트림에 대해 버퍼링 작업을 수정해서 명령 실행
stty	터미널 라인 설정을 변경 또는 출력
sum	지정된 각 파일에 대한 체크섬 및 블록 수 출력
sync	파일 시스템 버퍼를 비움; 변경된 블록들을 디스크에 기록하고 수퍼 블록을 업데이트함
tac	지정된 파일들을 역순으로 연결
tail	지정된 각 파일의 마지막 열 줄(또는 지정된 줄 수) 출력
tee	표준 입력을 읽어서 표준 출력과 지정된 파일로 동시에 기록
test	값을 비교하고 파일 형식 확인
timeout	시간 제한과 함께 명령 실행
touch	파일 타임스탬프 변경; 주어진 파일의 접근 및 수정 시간을 현재 시간으로 설정; 존재하지 않는 파일이면 시간을 0으로 생성
tr	표준 입력에서 지정된 문자를 치환, 압축 및 삭제
true	아무 작업도 수행하지 않음; 항상 성공을 나타내는 상태 코드와 함께 종료됨
truncate	지정된 크기로 파일 축소 또는 확장
tsort	토폴로지 정렬을 수행; 지정된 파일의 부분 순서에 따라 완전히 정렬된 목록을 작성
tty	표준 입력에 연결된 터미널의 파일 이름 출력
uname	시스템 정보 출력
unexpand	공백 문자를 탭 문자로 변환
uniq	중복된 행들을 하나만 제외하고 모두 제거
unlink	지정된 파일 삭제
users	현재 로그온한 유저의 이름 출력
vdir	ls -l와 동일
wc	지정된 파일의 행, 단어 및 바이트 수 출력. 둘 이상의 파일이 주어진 경우 행 갯수의 합계도 함께 출력
who	로그온 한 유저 출력
whoami	현재 실행 중인 유저 ID에 해당하는 유저 이름 출력
yes	종료될 때까지 “y” 또는 지정된 문자열을 반복해서 출력
libstdbuf	stdbuf에서 사용하는 라이브러리

6.55. Check-0.14.0

Check는 C를 위한 유닛 테스트 프레임워크이다.

예상 빌드 시간: 0.1 SBU (테스트 포함 약 3.5 SBU)
필요 디스크 공간: 13 MB

6.55.1. Check 설치

Check 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 빌드한다:

```
make
```

이제 컴파일이 끝났다. 테스트 스위트를 실행하려면, 다음 명령을 실행하라:

```
make check
```

Check 테스트 스위트는 비교적 긴 시간(최대 4 SBU)이 소요될 수 있다는 점을 참고하라.

패키지를 설치하고 스크립트를 수정한다:

```
make docdir=/usr/share/doc/check-0.14.0 install &&
sed -i '1 s/tools/usr/' /usr/bin/checkmk
```

6.55.2. Check의 내용

설치된 프로그램: checkmk
설치된 라이브러리: libcheck.{a,so}

요약

checkmk	Check 유닛 테스트 프레임워크에 사용하기 위한 C 유닛 테스트를 생성하는 Awk 스크립트
libcheck.{a,so}	테스트 프로그램에서 Check를 호출할 수 있는 함수 포함

6.56. Diffutils-3.7

Diffutils 패키지는 파일 또는 디렉토리 간의 차이를 보여주는 프로그램을 포함하고 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 36 MB

6.56.1. Diffutils 설치

Diffutils 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.56.2. Diffutils의 내용

설치된 프로그램들: cmp, diff, diff3, sdiff

요약

cmp 두 개의 파일을 비교하여 서로 다른 바이트의 유무를 출력
diff 두 파일 또는 디렉토리를 비교해서 파일의 행이 다른 부분을 출력
diff3 세 개의 파일을 한 줄씩 비교
sdiff 두 파일을 병합하고 대화식으로 결과 출력

6.57. Gawk-5.0.1

Gawk 패키지에는 텍스트 파일을 조작하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.4 SBU
필요 디스크 공간: 47 MB

6.57.1. Gawk 설치

우선 필요하지 않은 파일들이 설치되지 않도록 한다:

```
sed -i 's/extras//' Makefile.in
```

Gawk 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

필요하다면 문서를 설치하라:

```
mkdir -v /usr/share/doc/gawk-5.0.1
cp    -v doc/{awkforai.txt,*.{eps,pdf,jpg}} /usr/share/doc/gawk-5.0.1
```

6.57.2. Gawk의 내용

설치된 프로그램들: awk (gawk로 링크), gawk, awk-5.0.1
설치된 라이브러리들: filefuncs.so, fnmatch.so, fork.so, inplace.so, intdiv.so, ordchr.so, readdir.so, readfile.so, revoutput.so, revtwoway.so, rwarray.so, time.so (/usr/lib/gawk의 모든 파일)
설치된 디렉토리들: /usr/lib/gawk, /usr/libexec/awk, /usr/share/awk, /usr/share/doc/gawk-5.0.1

요약

awk	gawk로의 링크
gawk	텍스트 파일을 조작하는 프로그램; awk의 GNU 도구이다.
gawk-5.0.1	gawk로의 하드 링크

6.58. Findutils-4.7.0

Findutils 패키지는 파일을 찾는 프로그램을 포함하고 있다. 디렉토리 트리를 재귀적으로 검색하고 데이터베이스를 작성, 유지관리 및 검색(보통은 재귀 검색보다 빠르지만 데이터베이스가 최근에 업데이트되지 않았다면 신뢰할 수 없다)하기 위한 프로그램이다.

예상 빌드 시간: 0.7 SBU
필요 디스크 공간: 57 MB

6.58.1. Findutils 설치

Findutils 컴파일을 준비한다:

```
./configure --prefix=/usr --localstatedir=/var/lib/locate
```

Configure 옵션의 의미:

--localstatedir
locate 데이터베이스의 위치를 FHS를 준수하는 /var/lib/locate로 변경한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

chroot 환경에서 두 테스트들이 실패하는 것으로 알려져 있다: sv-bug-54171.old-O3과 sv-bug-54171.new-O3.

패키지를 설치한다:

```
make install
```

LFS-Bootscripts 패키지의 일부 스크립트는 find가 필요하다. 부팅 과정 초기에는 /usr을 사용할 수 없으므로 이 프로그램은 root 파티션에 있어야 한다. updatedb 스크립트도 수정해서 명시된 경로를 올바르게 고쳐야 한다:

```
mv -v /usr/bin/find /bin
sed -i 's|find:=${BINDIR}|find:=/bin|' /usr/bin/updatedb
```

6.58.2. Findutils의 내용

설치된 프로그램들: find, locate, updatedb, xargs
설치된 디렉토리: /var/lib/locate

요약

find	지정된 디렉토리 트리에서 지정된 기준과 일치하는 파일을 검색
locate	파일 이름의 데이터베이스를 검색하여 주어진 문자열을 포함하거나 주어진 패턴에 일치하는 이름을 출력
updatedb	locate 데이터베이스를 업데이트함; 전체 파일 시스템을 스캔하고 찾은 모든 파일 이름을 데이터베이스에 저장

xargs	지정된 명령을 여러 파일들에 적용하는 데 사용

6.59. Groff-1.22.4

Groff 패키지는 텍스트를 처리하고 서식을 지정하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.5 SBU
필요 디스크 공간: 95 MB

6.59.1. Groff 설치

Groff는 환경 변수 PAGE의 값이 기본 용지 크기여야 한다. 미국의 사용자에게는 PAGE=letter가 적합하다. 다른 곳에서는 PAGE=A4가 더 적합할 것이다. 컴파일 중에 기본 용지 크기가 설정되지만 나중에 /etc/papersize 파일에서 “A4”나 “letter”로 덮어쓸 수 있다.

Groff 컴파일을 준비한다:

```
PAGE=<paper_size> ./configure --prefix=/usr
```

이 패키지는 병렬 빌드를 지원하지 않는다. 패키지를 컴파일한다:

```
make -j1
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

6.59.2. Groff의 내용

설치된 프로그램들: addftinfo, afmtodit, chem, eqn, eqn2graph, gdiffmk, glilypond, gperl, gpinyin, grap2graph, grn, grodvi, groff, groffer, grog, grolbp, grolj4, gropdf, grops, grotty, hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pdfmom, pdfroff, pfbtops, pic, pic2graph, post-grohtml, preconv, pre-grohtml, refer, roff2dvi, roff2html, roff2pdf, roff2ps, roff2text, roff2x, soelim, tbl, tfmtodit, troff
설치된 디렉토리들: /usr/lib/groff와 /usr/share/doc/groff-1.22.4, /usr/share/groff

요약

addftinfo	Troff 글꼴 파일을 읽고 groff 시스템에 사용되는 일부 추가 글꼴 메트릭 정보를 추가
afmtodit	groff와 grops에 사용할 글꼴 파일 생성
chem	화학 구조 다이어그램 생성을 위한 Groff 전처리기
eqn	Troff 입력 파일에 포함된 방정식에 관한 설명을 troff가 인식할 수 있는 명령으로 컴파일
eqn2graph	Troff EQN(방정식, equation)을 잘라낸 이미지로 변환
gdiffmk	Groff/nroff/troff 파일 간의 차이 표시
glilypond	Lilypond 언어로 쓰인 악보를 groff 언어로 변형
gperl	Groff 파일에 perl 코드를 추가할 수 있는 groff용 전처리기
gpinyin	한어 병음 Pinyin을 groff 파일에 추가할 수 있는 groff 전처리기.
grap2graph	grap 다이어그램을 잘라낸 비트맵 이미지로 변환

grn	Gremlin 파일을 위한 groff 전처리기
grodvi	TeX dvi 포맷을 만드는 groff용 드라이버
groff	Groff 문서 서식 편집 프로그램의 프론트 엔드; 일반적으로 troff 프로그램과 선택된 장치에 적합한 후처리기를 실행
groffer	X와 tty 터미널에 groff 파일과 man 페이지 출력
grog	파일을 읽고 인쇄에 필요한 groff 옵션(-e, -man, -me, -mm, -ms, -p, -s, -t)을 추측해서 해당 옵션들을 포함한 groff 명령을 출력
grolbp	캐논 CAPSL 프린터용 groff 드라이버(LBP-4및 LBP-8 시리즈 레이저 프린터)
grolj4	HP LaserJet 4 프린터에 적합한 PCL5 형식의 출력을 생성하는 groff용 드라이버
gropdf	GNU troff의 출력을 PDF로 변환
grops	GNU troff의 출력을 PostScript로 변환
grotty	GNU troff의 출력을 타자기와 같은 장치에 적합한 형태로 변환
hpftodit	HP 태그 글꼴 메트릭 파일로부터 groff -Tlj4와 함께 사용할 글꼴 파일 생성
indxbib	refer, lookbib, lkbib에 사용하기 위해, 지정된 파일을 사용하여 서지 데이터베이스에 대한 반전된 색인을 작성
lkbib	서지 데이터베이스에서 지정된 키가 포함된 참고 문헌을 찾고 검색된 모든 참고 문헌을 출력
lookbib	표준 오류에 프롬프트를 출력(표준 입력이 터미널이 아닌 경우)하고, 표준 입력으로부터 키워드들을 읽어들인 다음, 지정된 파일에서 해당 키워드를 포함하는 참조문을 검색해서 표준 출력으로 출력한다. 입력이 끝날 때까지 이 과정을 반복함.
mmroff	groff를 위한 간단한 전처리기
neqn	미국 정보 교환 표준 부호(American Standard Code for Information Interchange, ASCII) 출력으로 방정식 서식화
nroff	groff를 사용하여 nroff 명령을 에뮬레이트하는 스크립트
pdfmom	mom 매크로로 서식화된 파일로부터 PDF 문서를 쉽게 만들 수 있는 groff용 래퍼.
pdfroff	Groff를 사용하여 pdf 문서 작성
pfbtops	.pfb 형식의 PostScript 글꼴을 ASCII로 변환
pic	troff 또는 TeX 입력 파일에 포함된 사진에 대한 설명을 TeX나 troff가 인식 가능한 명령으로 컴파일
pic2graph	PIC 다이어그램을 잘라낸 이미지로 변환
post-grohtml	GNU troff의 출력을 HTML로 변환
preconv	입력 파일의 인코딩을 GNU troff가 인식하는 것으로 변환
pre-grohtml	GNU troff의 출력을 HTML로 변환
refer	인용문으로 해석되는 .[와 .] 사이의 행과, 인용문을 어떻게 처리할지에 대한 명령으로 해석되는 .R1과 .R2 사이의 행을 제외하고, 파일의 내용을 표준 출력에 복사한다.
roff2dvi	Roff 파일을 DVI 형식으로 변환
roff2html	Roff 파일을 html 형식으로 변환
roff2pdf	Roff 파일을 pdf 형식으로 변환
roff2ps	Roff 파일을 ps 형식으로 변환

roff2text	Roff 파일을 텍스트 형식으로 변환
roff2x	Roff 파일을 다른 형식으로 변환
soelim	파일을 읽어들여서 언급된 file의 내용으로 .so file 형식의 행을 대체함
tbl	Troff 입력 파일에 포함된 테이블에 대한 설명을 troff가 인식하는 명령으로 컴파일
tfmtodit	groff -Tdvi와 함께 사용할 글꼴 파일 작성
troff	유닉스 troff와 매우 잘 호환됨; 일반적으로groff 명령을 사용해서 호출되어야 한다. 이 명령은 적절한 순서와 적절한 옵션으로 전처리기와 후처리기를 실행한다

6.60. GRUB-2.04

GRUB 패키지에는 GRand Unified Bootloader가 포함되어 있다.

예상 빌드 시간: 0.8 SBU
필요 디스크 공간: 161 MB

6.60.1. GRUB 설치

GRUB 컴파일을 준비한다:

```
./configure --prefix=/usr        ₩
            --sbindir=/sbin      ₩
            --sysconfdir=/etc    ₩
            --disable-efiemu     ₩
            --disable-werror
```

새 Configure 옵션들의 의미:

--disable-werror
보다 최신 버전의 Flex에서 알려진 경고 사항들을 기반으로 빌드를 한다.

--disable-efiemu
LFS에 필요하지 않은 기능과 테스트 프로그램을 비활성화하여 빌드할 대상을 최소화한다.

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
mv -v /etc/bash_completion.d/grub /usr/share/bash-completion/completions
```

GRUB으로 LFS 시스템을 부팅하는 방법은 8.4절. “GRUB을 사용하여 부팅 프로세스 설정”에서 설명한다.

6.60.2. GRUB의 내용

설치된 프로그램들: grub-bios-setup, grub-editenv, grub-file, grub-fstest, grub-glue-efi, grub-install, grub-kbdcomp, grub-macbless, grub-menulst2cfg, grub-mkconfig, grub-mkimage, grub-mklayout, grub-mknetdir, grub-mkpasswd-pbkdf2, grub-mkrelpath, grub-mkrescue, grub-mkstandalone, grub-ofpathname, grub-probe, grub-reboot, grub-render-label, grub-script-check, grub-set-default, grub-sparc64-setup, grub-syslinux2cfg
설치된 디렉토리들: /usr/lib/grub, /etc/grub.d, /usr/share/grub, /boot/grub (grub-install이 처음 실행될 때)

요약

grub-bios-setup	Grub-install을 위한 헬퍼 프로그램
grub-editenv	환경 블록을 편집하는 도구

grub-file	FILE이 지정된 유형인지 확인
grub-fstest	파일 시스템 드라이버를 디버그하는 도구
grub-glue-efi	ia32 및 amd64 EFI 이미지를 처리하고 Apple 포맷에 따라 합침.
grub-install	드라이브에 GRUB 설치
grub-kbdcomp	xkb 레이아웃을 GRUB에서 인식되는 레이아웃으로 변환하는 스크립트
grub-macbless	HFS나 HFS+ 파일에 대한 Mac 스타일 bless
grub-menulst2cfg	GRUB 2와 함께 사용할 GRUB 레거시 menu.lst를 grub.cfg로 변환
grub-mkconfig	GRUB 구성 파일 생성
grub-mkimage	GRUB의 부트 이미지 제작
grub-mklayout	GRUB 키보드 레이아웃 파일 생성
grub-mknetdir	GRUB Netboot 디렉토리 준비
grub-mkpasswd-pbkdf2	부팅 메뉴에서 사용할 암호화된 PBKDF2 암호 생성
grub-mkrelpath	root를 기준으로 시스템 경로명 생성
grub-mkrescue	플로피 디스크 또는 CDROM/DVD에 적합한 GRUB의 부트 이미지 제작
grub-mkstandalone	독립 실행형 이미지 생성
grub-ofpathname	GRUB 장치의 경로를 출력하는 헬퍼 프로그램
grub-probe	지정된 경로나 장치에 대한 장치 정보 조사
grub-reboot	다음 부팅에 대해서만 GRUB에 대한 기본 부팅 항목 설정
grub-render-label	Apple Mac용 Apple .disk_label 렌더링
grub-script-check	GRUB 구성 스크립트에서 구문 오류 확인
grub-set-default	GRUB의 기본 부팅 항목 설정
grub-sparc64-setup	grub-setup을 위한 헬퍼 프로그램
grub-syslinux2cfg	syslinux 구성 파일을 grub.cfg 형식으로 변환

6.61. Less-551

Less 패키지에는 텍스트 파일 뷰어가 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 4.1 MB

6.61.1. Less 설치

Less 컴파일을 준비한다:

```
./configure --prefix=/usr --sysconfdir=/etc
```

Configure 옵션의 의미:

--sysconfdir=/etc
패키지에 의해 생성된 프로그램이 /etc에서 구성 파일을 찾도록 지시한다.

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

6.61.2. Less의 내용

설치된 프로그램들: less, lessecho, lesskey

요약

less	파일 뷰어 또는 호출기; 지정된 파일의 내용을 출력하여 사용자가 스크롤하고 문자열을 찾고 표시한 곳으로 이동할 수 있음
lessecho	유닉스 시스템의 파일 이름에서 * 및 ?와 같은 메타 문자를 확장하는 데 필요
lesskey	less의 키 바인딩을 지정하기 위해 사용

6.62. Gzip-1.10

Gzip 패키지에는 파일을 압축하고 해제하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 20 MB

6.62.1. Gzip 설치

Gzip 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

LFS 환경에서 두 테스트가 실패하는 것으로 알려져 있다: help-version, zmore.

패키지를 설치한다:

```
make install
```

root 파일 시스템에 있어야 하는 프로그램을 이동한다:

```
mv -v /usr/bin/gzip /bin
```

6.62.2. Gzip의 내용

설치된 프로그램들: gunzip, gzexe, gzip, uncompress (gunzip과 하드 링크), zcat, zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore, znew

요약

gunzip	gzip으로 압축된 파일을 압축 해제
gzexe	자체 압축 해제 실행 파일 생성
gzip	Lempel-Ziv (LZ77) 코딩을 사용하여 지정된 파일 압축
uncompress	압축된 파일의 압축 해제
zcat	지정된 gzip 압축 파일을 표준 출력으로 압축 해제
zcmp	gzip 압축 파일에서 cmp 실행
zdiff	gzip 압축 파일에서 diff 실행
zegrep	gzip 압축 파일에서 egrep 실행
zfgrep	gzip 압축 파일에서 fgrep 실행
zforce	gzip으로 압축된 모든 파일에 .gz 확장자를 강제로 붙여 gzip이 다시 압축하지 않도록 함; 파일 전송 중에 파일 이름이 잘리는 경우 유용함
zgrep	gzip 압축 파일에서 grep 실행

zless	gzip 압축 파일에서 less 실행
zmore	gzip 압축 파일에서 more 실행
znew	compress 형식에서 gzip 형식으로 파일을 다시 압축—.Z에서 .gz로

6.63. Zstd-1.4.4

Zstandard는 실시간 압축 알고리즘으로 높은 압축률을 제공한다. 매우 빠른 디코더의 도움으로 매우 광범위한 압축/속도 절충 기능을 지원한다.

예상 빌드 시간: 0.7 SBU
필요 디스크 공간: 16 MB

6.63.1. Zstd 설치

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make prefix=/usr install
```

정적 라이브러리를 제거하고 공유 라이브러리를 /lib로 이동하라. 또한 /usr/lib의 .so 파일을 재생성해야 한다:

```
rm -v /usr/lib/libzstd.a
mv -v /usr/lib/libzstd.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libzstd.so) /usr/lib/libzstd.so
```

6.63.2. Zstd의 내용

설치된 프로그램들: zstd, zstdcat (zstd로 링크), zstdgrep, zstdless, zstdmt (zstd로 링크), unzstd (zstd로 링크)
설치된 라이브러리: libzstd.so

요약

zstd	ZSTD 형식을 사용하여 파일 압축 또는 압축 해제
zstdgrep	ZSTD 압축 파일에서 grep 실행
zstdless	ZSTD 압축 파일에서 less 실행
libzstd	ZSTD 알고리즘을 사용하여 무손실 데이터 압축을 구현하는 라이브러리

6.64. IPRoute2-5.5.0

IPRoute2 패키지에는 기본 및 고급 IPV4 기반 네트워킹 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 14 MB

6.64.1. IPRoute2 설치

이 패키지에 포함된 arpd 프로그램은 LFS에 설치되지 않은 버클리 DB가 필요하기 때문에 빌드되지 않는다. 그러나 arpd의 디렉토리와 man 페이지는 여전히 설치될 것이다. 아래 명령을 실행하여 설치를 방지하라. arpd 바이너리가 필요한 경우, 버클리 DB를 컴파일하기 위한 지침은 BLFS 책의 http://www.linuxfromscratch.org/blfs/view/9.1/server/databases.html#db에서 확인할 수 있다.

```
sed -i /ARPD/d Makefile
rm -fv man/man8/arpd.8
```

http://www.linuxfromscratch.org/blfs/view/9.1/postlfs/iptables.html이 필요한 두 모듈의 빌드도 비활성화해야 한다.

```
sed -i 's/.m_ipt.o//' tc/Makefile
```

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make DOCDIR=/usr/share/doc/iproute2-5.5.0 install
```

6.64.2. IPRoute2의 내용

설치된 프로그램들: bridge, ctstat (lnstat로 링크), genl, ifcfg, ifstat, ip, lnstat, nstat, routef, routel, rtacct, rtmon, rtpr, rtstat (lnstat로 링크), ss, tc
설치된 디렉토리들: /etc/iproute2, /usr/lib/tc, /usr/share/doc/iproute2-5.5.0,

요약

bridge	네트워크 브릿지 구성
ctstat	연결 상태 유틸리티
genl	일반 netlink 유틸리티 프론트엔드
ifcfg	ip 명령의 셸 스크립트 래퍼 [참고: http://www.skbuff.net/iputils/에 있는 iputils 패키지의 arping 및 rdisk 프로그램이 필요함.]
ifstat	인터페이스별 발신 및 수신 패킷 양을 포함한 인터페이스 통계 표시
ip	주 실행 파일. 이것은 몇 가지 다른 기능을 갖고 있다: ip link <device>는 사용자가 장치 상태를 보고 변경할 수 있다. ip addr는 사용자가 주소와 그 속성을 보고, 새 주소를 추가하거나 이전 주소를 삭제할 수 있다. ip neighbor는 사용자가 이웃 바인딩 및 해당 속성을 보고, 새 이웃 항목을 추가하거나 이전 항목을 삭제할 수 있다.

	ip rule는 라우팅 정책을 보고 변경할 수 있다 ip route는 사용자가 라우팅 테이블을 보고 라우팅 테이블 규칙을 변경할 수 있다. ip tunnel은 사용자가 IP 터널과 해당 속성을 보고 변경할 수 있다. ip maddr은 사용자가 멀티캐스트 주소와 그들의 속성을 보고 그것들을 변경할 수 있게 해준다. ip mroute는 사용자가 멀티캐스트 라우팅을 설정, 변경 또는 삭제할 수 있다. ip monitor는 사용자가 기기, 주소 및 경로의 상태를 지속적으로 모니터링할 수 있다.
lnstat	리눅스 네트워크 통계 제공; 이전의 rtstat 프로그램보다 일반적이고 완전한 기능을 가진 대체품
nstat	네트워크 통계 표시
routef	ip route의 구성 요소. 라우팅 테이블을 비우는 용도
routel	ip route의 구성 요소. 라우팅 테이블을 나열하는 용도
rtacct	/proc/net/rt_acct의 내용 표시
rtmon	경로(route) 모니터링 유틸리티
rtpr	ip -o 출력을 다시 읽을 수 있는 형식으로 변환
rtstat	경로 상태 유틸리티
ss	netstat 명령어와 유사; 활성 상태인 연결을 표시
tc	트래픽 제어 실행 파일; Quality Of Service (QOS)와 Class Of Service (COS) 구현용이다 tc qdisc는 사용자가 대기열 규칙(queuing discipline)을 설정할 수 있다. tc class는 사용자가 대기열 스케줄링(queuing discipline scheduling)에 따라 클래스를 설정할 수 있다. tc estimator는 사용자가 네트워크로의 네트워크 플로우를 추정할 수 있다. tc filter는 사용자가 QOS/COS 패킷 필터링을 설정할 수 있다. tc policy는 사용자가 QOS/COS 정책을 설정할 수 있다.

6.65. Kbd-2.2.0

Kbd 패키지에는 키 테이블 파일, 콘솔 글꼴 및 키보드 유틸리티가 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 36 MB

6.65.1. Kbd 설치

Backspace와 Delete 키의 동작은 Kbd 패키지의 키맵에서 일관되지 못하다. 다음 패치는 i386 키맵에 대해 이 문제를 수정한다:

```
patch -Np1 -i ../kbd-2.2.0-backspace-1.patch
```

패치를 하고나면 Backspace 키는 코드 127로 문자를 생성하며 Delete 키는 잘 알려진 이스케이프 시퀀스(escape sequence)를 생성한다.

man 페이지와 함께 불필요한 resizecons 프로그램을 삭제한다(이 프로그램은 비디오 모드 파일을 제공하기 위해 현존하지 않는 svgalib이 필요하다 - 보통 콘솔 사이즈를 적절히 바꾸기 위해서는 setfont를 사용).

```
sed -i 's/₩(RESIZECONS_PROGS=₩)yes/₩1no/g' configure
sed -i 's/resizecons.8 //' docs/man/man8/Makefile.in
```

Kbd 컴파일을 준비한다:

```
PKG_CONFIG_PATH=/tools/lib/pkgconfig ./configure --prefix=/usr --disable-vlock
```

Configure 옵션의 의미:

--disable-vlock
chroot 환경에서 사용할 수 없는 PAM 라이브러리가 필요하므로, vlock 유틸리티가 빌드되지 않도록 한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

참고

일부 언어(예: 벨로루시어)의 경우 Kbd 패키지에서는 ISO-8859-5 인코딩을 사용하는 키맵이 제공되지 않고, CP1251 키맵이 사용된다. 이러한 언어의 사용자는 별도로 키맵을 다운로드 받아야 한다.

필요하다면 문서를 설치하라:

```
mkdir -v       /usr/share/doc/kbd-2.2.0
cp -R -v docs/doc/* /usr/share/doc/kbd-2.2.0
```

6.65.2. Kbd의 내용

설치된 프로그램들: chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, kbdinfo, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (psfxtable로 링크), psfgettable (psfxtable로 링크), psfstriptable (psfxtable로 링크), psfxtable, setfont, setkeycodes, setleds, setmetamode, setvtrgb, showconsolefont, showkey, unicode_start, unicode_stop

설치된 디렉토리들: /usr/share/consolefonts, /usr/share/consoletrans, /usr/share/keymaps, /usr/share/doc/kbd-2.2.0, /usr/share/unimaps

요약

chvt	가상 터미널 변경
deallocvt	사용되지 않는 가상 터미널의 할당 해제
dumpkeys	키보드 변환 테이블 덤프
fgconsole	활성 가상 터미널 수 출력
getkeycodes	커널 스캔코드-키코드 매핑 테이블 출력
kbdinfo	콘솔 상태에 대한 정보를 가져옴
kbd_mode	키보드 모드 출력 또는 설정
kbdrate	키보드 반복 및 지연 속도 설정
loadkeys	키보드 변환 테이블 로드
loadunimap	커널 유니코드-글꼴 매핑 테이블 로드
mapscrn	사용자 정의 출력 문자 매핑 테이블을 콘솔 드라이버에 로드하는 데 사용된 오래된 프로그램; 이제 setfont가 대신함.
openvt	새 가상 터미널(VT)에서 프로그램 시작
psfaddtable	콘솔 글꼴에 유니코드 문자 테이블 추가
psfgettable	콘솔 글꼴에서 내장된 유니코드 문자 테이블 추출
psfstriptable	콘솔 글꼴에서 내장된 유니코드 문자 테이블 제거
psfxtable	콘솔 글꼴에 대한 유니코드 문자 테이블 처리
setfont	콘솔에서 Enhanced Graphic Adapter (EGA) 및 Video Graphics Array (VGA) 글꼴 변경
setkeycodes	커널 스캔코드-키코드 매핑 테이블 항목 로드; 키보드에 특이한 키가 있는 경우 유용함
setleds	키보드 플래그 및 발광 다이오드(LED) 설정
setmetamode	키보드 메타 키 처리 정의
setvtrgb	모든 가상 터미널의 콘솔 컬러맵 설정
showconsolefont	현재 EGA/VGA 콘솔 화면 글꼴 표시
showkey	키보드에서 누른 키의 스캔 코드, 키 코드 및 ASCII 코드를 출력
unicode_start	키보드와 콘솔을 UNICODE 모드로 설정 [키맵 파일이 ISO-8859-1 인코딩이 아닌 이상 이 프로그램을 사용하지 않도록 하라. 다른 인코딩의 경우 이 유틸리티는 잘못된 결과를 낳는다.]
unicode_stop	키보드 및 콘솔의 UNICODE 모드 중지

6.66. Libpipeline-1.5.2

Libpipeline 패키지에는 유연하고 편리하게 하위 프로세스의 파이프라인을 조작할 수 있는 라이브러리를 포함하고 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 9.2 MB

6.66.1. Libpipeline 설치

Libpipeline 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.66.2. Libpipeline의 내용

설치된 라이브러리: libpipeline.so

요약

libpipeline 하위 프로세스 간의 파이프라인을 안전하게 구성하는 라이브러리

6.67. Make-4.3

Make 패키지는 패키지를 컴파일하기 위한 프로그램을 포함하고 있다.

예상 빌드 시간: 0.5 SBU
필요 디스크 공간: 16 MB

6.67.1. Make 설치

Make 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

테스트 스위트는 지원하는 Perl 파일들이 어디에 있는지 알아야 한다. 따라서 환경 변수를 사용해서 위치를 전달한다. 결과를 테스트하려면 다음을 실행하라:

```
make PERL5LIB=$PWD/tests/ check
```

패키지를 설치한다:

```
make install
```

6.67.2. Make의 내용

설치된 프로그램: make

요약

make 어떤 패키지를 (재)컴파일해야 하는지 자동으로 결정한 다음 관련 명령들을 실행

6.68. Patch-2.7.6

Patch 패키지에는 일반적으로 diff 프로그램으로 만든 “patch” 파일을 적용하여 파일을 수정하거나 만드는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 13 MB

6.68.1. Patch 설치

Patch 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.68.2. Patch의 내용

설치된 프로그램: patch

요약

patch 패치 파일에 따라 파일 수정 [패치 파일은 보통 diff 프로그램으로 작성된 차이점 목록이다. 이러한 차이점을 원본 파일에 적용하면 patch가 패치된 버전을 생성한다.]

6.69. Man-DB-2.9.0

Man-DB 패키지에는 Man 페이지를 검색하고 읽을 수 있는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.5 SBU
필요 디스크 공간: 40 MB

6.69.1. Man-DB 설치

Man-DB 컴파일을 준비한다:

```
./configure --prefix=/usr                        ₩
            --docdir=/usr/share/doc/man-db-2.9.0 ₩
            --sysconfdir=/etc                    ₩
            --disable-setuid                     ₩
            --enable-cache-owner=bin             ₩
            --with-browser=/usr/bin/lynx         ₩
            --with-vgrind=/usr/bin/vgrind        ₩
            --with-grap=/usr/bin/grap            ₩
            --with-systemdtmpfilesdir=           ₩
            --with-systemdsystemunitdir=
```

Configure 옵션들의 의미:

--disable-setuid
man 프로그램 setuid를 man 유저로 만들지 않도록 한다.

--enable-cache-owner=bin
시스템 전체 캐시 파일의 소유권을 bin에게 부여한다.

--with-...
이 세 매개 변수들은 몇 가지 기본 프로그램을 설정한다. lynx는 텍스트 기반 웹 브라우저(설치 절차는 BLFS를 참고)이고, vgrind는 프로그램 소스를 Groff 입력으로 변환하며, grap은 Groff 문서에 그래프를 삽입하는 데에 유용하다. vgrind와 grap 프로그램은 보통은 매뉴얼 페이지를 보는 데 필요하지 않다. LFS나 BLFS에서 다루진 않지만, 원한다면 LFS를 완료하고 스스로 설치할 수 있을 것이다.

--with-systemd...
불필요한 systemd 디렉토리 및 파일 설치를 방지한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

6.69.2. LFS의 비-영어권 매뉴얼 페이지

다음 표는 /usr/share/man/<ll>에 설치된 Man-DB의 매뉴얼 페이지들이 인코딩될 수 있는 문자 세트들을 보여준다. 이 외에도, Man-DB는 해당 디렉토리에 설치된 매뉴얼 페이지들이 UTF-8로 인코딩되어 있는지 정확하게 확인한다.

표 6.1. 기존 8비트 매뉴얼 페이지의 문자 인코딩

언어 (코드)	인코딩	언어 (코드)	인코딩
덴마크어 (da)	ISO-8859-1	크로아티아어 (hr)	ISO-8859-2
독일어 (de)	ISO-8859-1	헝가리어 (hu)	ISO-8859-2
영어 (en)	ISO-8859-1	일본어 (ja)	EUC-JP
스페인어 (es)	ISO-8859-1	한국어 (ko)	EUC-KR
에스토니아어 (et)	ISO-8859-1	리투아니아어 (lt)	ISO-8859-13
핀란드어 (fi)	ISO-8859-1	라트비아어 (lv)	ISO-8859-13
프랑스어 (fr)	ISO-8859-1	마케도니아어 (mk)	ISO-8859-5
아일랜드어 (ga)	ISO-8859-1	폴란드어 (pl)	ISO-8859-2
갈리시아어 (gl)	ISO-8859-1	루마니아어 (ro)	ISO-8859-2
인도네시아어 (id)	ISO-8859-1	러시아어 (ru)	KOI8-R
아이슬란드어 (is)	ISO-8859-1	슬로바키아어 (sk)	ISO-8859-2
이탈리아어 (it)	ISO-8859-1	슬로베니아어 (sl)	ISO-8859-2
노르웨이어 보크몰 (nb)	ISO-8859-1	세르비아어 로마자 (sr@latin)	ISO-8859-2
네덜란드어 (nl)	ISO-8859-1	세르비아어 (sr)	ISO-8859-5
노르웨이어 뉘노르스크 (nn)	ISO-8859-1	터키어 (tr)	ISO-8859-9
노르웨이어 (no)	ISO-8859-1	우크라이나어 (uk)	KOI8-U
포르투갈어 (pt)	ISO-8859-1	베트남어 (vi)	TCVN5712-1
스웨덴어 (sv)	ISO-8859-1	중국어 간체 (zh_CN)	GBK
벨라루스어 (be)	CP1251	싱가폴, 중국어 간체 (zh_SG)	GBK
불가리아어 (bg)	CP1251	홍콩, 중국어 정체 (zh_HK)	BIG5HKSCS
체코어 (cs)	ISO-8859-2	중국어 정체 (zh_TW)	BIG5
그리스어 (el)	ISO-8859-7		

참고

목록에 없는 언어의 매뉴얼 페이지는 지원되지 않는다.

6.69.3. Man-DB의 내용

설치된 프로그램들: accessdb, apropos (whatis로 링크), catman, lexgrog, man, mandb, manpath, whatis
설치된 라이브러리들: libman.so와 libmandb.so (/usr/lib/man-db의 두 가지)
설치된 디렉토리들: /usr/lib/man-db, /usr/libexec/man-db, /usr/share/doc/man-db-2.9.0

요약

accessdb 사람이 읽을 수 있는 형식으로 whatis 데이터베이스 내용 덤프

apropos	whatis 데이터베이스를 검색해서 지정된 문자열을 포함하는 시스템 명령에 대한 간단한 설명 표시
catman	사전 서식화(pre-formatted)된 매뉴얼 페이지 생성 또는 업데이트
lexgrog	지정된 매뉴얼 페이지에 대한 한 줄 요약 정보 출력
man	요청된 매뉴얼 페이지 서식화 및 출력
mandb	whatis 데이터베이스 작성 또는 업데이트
manpath	man.conf와 유저의 설정을 기반으로 $MANPATH나 $MANPATH가 설정되지 않은 경우 적절한 검색 경로의 값을 표시
whatis	whatis 데이터베이스를 검색해서 지정된 키워드를 포함하는 시스템 명령에 대한 간단한 설명을 별도의 단어로 출력
libman	man에 대한 런타임 지원을 포함
libmandb	man에 대한 런타임 지원을 포함

6.70. Tar-1.32

Tar 패키지에는 아카이빙(archiving) 프로그램을 포함하고 있다.

예상 빌드 시간: 2.0 SBU
필요 디스크 공간: 45 MB

6.70.1. Tar 설치

Tar 컴파일을 준비한다:

```
FORCE_UNSAFE_CONFIGURE=1  ₩
./configure --prefix=/usr ₩
            --bindir=/bin
```

Configure 옵션들의 의미:

FORCE_UNSAFE_CONFIGURE=1
mknod에 대한 테스트를 root 유저로 실행하도록 강제한다. root 유저로 이 테스트를 실행하는 것은 보통 위험하다고 생각할 수 있지만, 부분적으로만 구축된 시스템에서 실행하는 것이므로 우려할 필요 없다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 (약 3 SBU), 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
make -C doc install-html docdir=/usr/share/doc/tar-1.32
```

6.70.2. Tar의 내용

설치된 프로그램: tar
설치된 디렉토리: /usr/share/doc/tar-1.32

요약

tar Tarball이라고도 하는 아카이브의 내용 작성, 추출 및 나열

6.71. Texinfo-6.7

Texinfo 패키지에는 info 페이지를 읽고, 쓰고, 변환하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.7 SBU
필요 디스크 공간: 116 MB

6.71.1. Texinfo 설치

Texinfo 컴파일을 준비한다:

```
./configure --prefix=/usr --disable-static
```

Configure 옵션들의 의미:

--disable-static
여기서 최상위 구성 스크립트로부터 인식되지 않은 옵션이라는 경고가 나오겠지만, XSParagraph의 구성 스크립트가 이 옵션을 인식해서 정적 XSParagraph.a가 /usr/lib/texinfo에 설치되는 것을 비활성화한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

선택 사항으로, TeX 설치에 속하는 구성 요소를 설치를 할 수 있다:

```
make TEXMF=/usr/share/texmf install-tex
```

Make 매개 변수의 의미:

TEXMF=/usr/share/texmf
예를 들어 TEXMF makefile 변수의 값이 TeX 트리의 최상위 디렉토리 위치이면, 이후에 TeX 패키지가 설치된다.

Info 문서 시스템은 메뉴 항목들의 목차를 일반 텍스트 파일에 저장한다. 그 파일들은 /usr/share/info/dir에 있다. 불행히도, 여러 패키지들의 Makefile의 간헐적인 문제때문에, 시스템에 설치된 info 페이지와 동기화되지 않을 수 있다. 만약 /usr/share/info/dir 파일을 재생성까지 해야하는 경우, 선택 사항인 다음 명령으로 수행할 수 있다:

```
pushd /usr/share/info
rm -v dir
for f in *
  do install-info $f dir 2>/dev/null
done
popd
```

6.71.2. Texinfo의 내용

설치된 프로그램들:	info, install-info, makeinfo (texi2any로 링크), pdftexi2dvi, pod2texi, texi2any, texi2dvi, texi2pdf, texindex
설치된 라이브러리:	MiscXS.so, Parsetexi.so, XSParagraph.so (/usr/lib/texinfo의 모든 것)
설치된 디렉토리들:	/usr/share/texinfo와 /usr/lib/texinfo

요약

info	Man 페이지와 유사한 info 페이지를 읽는 데 사용되지만, 단순히 사용 가능한 모든 명령줄 옵션을 설명하는 것보다 훨씬 더 자세한 경우가 많다 [예를 들면, man bison과 info bison를 비교해보라.]
install-info	Info 페이지를 설치하는 데 사용; info 인덱스 파일의 항목을 업데이트함
makeinfo	지정된 Texinfo 원본 문서를 info 페이지, 일반 텍스트 또는 HTML로 변환
pdftexi2dvi	지정된 Texinfo 문서를 PDF 파일로 서식화하는 데 사용
pod2texi	Pod를 Texinfo 형식으로 변환
texi2any	Texinfo 원본 문서를 다양한 다른 형식으로 변환
texi2dvi	지정된 Texinfo 문서를 인쇄할 수 있는 장치 독립적 파일로 서식화하는 데 사용
texi2pdf	지정된 Texinfo 문서를 PDF 파일로 서식화하는 데 사용
texindex	Texinfo 인덱스 파일을 정렬하는 데 사용

6.72. Vim-8.2.0190

Vim 패키지에는 강력한 텍스트 편집기가 포함되어 있다.

예상 빌드 시간: 1.7 SBU
필요 디스크 공간: 202 MB

Vim의 대체제

Emacs, Joe 또는 Nano와 같은 다른 편집기를 선호한다면 http://www.linuxfromscratch.org/blfs/view/9.1/postlfs/editors.html에서 제안된 설치 지침들을 참고하기 바란다.

6.72.1. Vim 설치

먼저 vimrc 설정 파일의 기본 위치를 /etc로 변경하라:

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
```

Vim 컴파일을 준비한다:

```
./configure --prefix=/usr
```

패키지를 컴파일한다:

```
make
```

테스트를 준비하기 위해 nobody 유저가 소스 트리에 쓰기 가능하도록 설정한다:

```
chown -Rv nobody .
```

이제 nobody 유저로 테스트를 실행하라:

```
su nobody -s /bin/bash -c "LANG=en_US.UTF-8 make -j1 test" &> vim-test.log
```

테스트 스위트는 많은 바이너리 데이터를 화면에 출력한다. 이로 인해 현재 터미널의 설정에 문제가 발생할 수 있다. 이 문제는 위에서 보인 바와 같이 출력을 로그 파일로 리다이렉팅해서 방지할 수 있다. 테스트가 성공적으로 끝나면 로그 파일 안에 "ALL DONE"이라는 문구가 표시된다.

패키지를 설치한다:

```
make install
```

많은 사람들이 vim 대신 vi을 입력한다. 사용자가 습관적으로 vi를 입력해도 vim이 실행되게 하려면, 실행 파일과 제공된 언어의 man 페이지에 심볼릭 링크를 생성하면 된다.:

```
ln -sv vim /usr/bin/vi
for L in  /usr/share/man/{,*/}man1/vim.1; do
    ln -sv vim.1 $(dirname $L)/vi.1
done
```

기본적으로 Vim의 문서들은 /usr/share/vim에 설치된다. 다음과 같이 심볼릭 링크를 만들어서 다른 패키지의 문서들처럼 /usr/share/doc/vim-8.2.0190를 통해 문서들에 접근할 수 있다:

```
ln -sv ../vim/vim82/doc /usr/share/doc/vim-8.2.0190
```

X 윈도우 시스템을 LFS 시스템에 설치하려는 경우, X 설치 후 Vim을 다시 컴파일해야 할 수도 있다. Vim은 X와 추가 라이브러리를 몇 개 설치해야 하는 GUI 버전이 같이 제공된다. 이 절차에 대한 자세한 내용은 Vim 설명서와 http://www.linuxfromscratch.org/blfs/view/9.1/postlfs/vim.html에서 BLFS 책의 Vim 설치 페이지를 참고하라.

6.72.2. Vim 설정

기본적으로 vim은 vi-비호환 모드로 실행된다. 이는 평소 다른 편집기를 사용하던 사용자들에게는 낯설 것이다. 아래에는 vi 대신 새로운 vim이 사용되고 있다는 것을 강조하기 위해 “nocompatible”(비호환) 설정이 명시되어 있다. 그리고 “compatible(호환)” 모드로 변경하려는 사용자들에게는 이 항목을 설정 파일의 첫 번째로 두어야 함을 상기시키고자 명시하였다. 이 설정은 다른 설정들을 변경하기 때문에 그 위에 덮어쓰기 위해선 반드시 이 설정 뒤에 기재되어야 한다. 다음을 실행하여 기본 vim 설정 파일을 생성하라:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc

" Ensure defaults are set before customizing settings, not after
source $VIMRUNTIME/defaults.vim
let skip_defaults_vim=1

set nocompatible
set backspace=2
set mouse=
syntax on
if (&term == "xterm") || (&term == "putty")
  set background=dark
endif

" End /etc/vimrc
EOF
```

set nocompatible 설정은 vim을 vi-compatible 방식보다 더 유용한 방식으로(기본값) 동작하게 한다. “no”를 지우면 오래된 vi 방식을 고수한다. set backspace=2 설정은 줄 바꿈, 자동 들여쓰기, 삽입의 시작 부분에서도 Backspace로 글자를 지울 수 있게 한다. syntax on 매개 변수는 Vim의 문법 강조 기능을 활성화한다. set mouse= 설정은 chroot나 원격 연결로 작업할 때 텍스트를 마우스로 적절히 붙여넣을 수 있다. 마지막으로, if 문의 set background=dark 설정은 일부 터미널 에뮬레이터의 배경색을 vim이 알맞게 인식하도록 한다. 즉, 저 프로그램들의 검정 배경에 쓰기 더 좋은 색 구성표를 적용한다.

다른 사용 가능한 옵션에 대한 문서는 다음 명령을 통해 얻을 수 있다:

```
vim -c ':options'
```

참고

기본적으로 Vim은 영어용 철자 파일만 설치한다. 원하는 언어의 철자 파일을 설치하려면 ftp://ftp.vim.org/pub/vim/runtime/spell/에서 *.spl과 언어 및 문자 인코딩용 *.sug 파일(선택 사항)을 다운로드하여 /usr/share/vim/vim82/spell/에 저장하라.

이러한 철자 파일을 사용하려면 /etc/vimrc에서 일부 설정이 필요하다. 예:

```
set spelllang=en,ru
set spell
```

더 많은 정보는 위의 URL에 있는 해당 README 파일을 참고하라.

6.72.3. Vim의 내용

설치된 프로그램들: ex (vim으로 링크), rview (vim으로 링크), rvim (vim으로 링크), vi (vim으로 링크), view (vim으로 링크), vim, vimdiff (vim으로 링크), vimtutor, xxd
설치된 디렉토리: /usr/share/vim

요약

ex	Vim을 ex 모드로 시작
rview	View의 제한된 버전; 셸 명령을 시작할 수 없으며 view를 일시 중지할 수 없음
rvim	Vim의 제한된 버전; 셸 명령을 시작할 수 없으며 vim을 일시 중지할 수 없음
vi	Vim으로의 링크
view	읽기 전용 모드에서 vim 시작
vim	편집기
vimdiff	Vim을 사용하여 한 파일의 둘이나 세 가지 버전을 편집하고 차이점을 표시
vimtutor	Vim의 기본적인 조작과 명령을 배울 수 있음
xxd	지정된 파일의 16진수 덤프 생성; 역으로도 가능하기 때문에 바이너리 패치에 사용할 수 있다

6.73. Procps-ng-3.3.15

Procps-ng 패키지에는 프로세스 모니터링 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU
필요 디스크 공간: 17 MB

6.73.1. Procps-ng 설치

procps-ng 컴파일을 준비한다:

```
./configure --prefix=/usr                        ₩
            --exec-prefix=                       ₩
            --libdir=/usr/lib                    ₩
            --docdir=/usr/share/doc/procps-ng-3.3.15 ₩
            --disable-static                     ₩
            --disable-kill
```

Configure 옵션들의 의미:

--disable-kill
Util-linux 패키지에 의해 설치될 kill 명령어 빌드를 비활성화한다.

패키지를 컴파일한다:

```
make
```

테스트 스위트는 LFS를 위한 몇 가지 수정이 필요하다. 스크립트가 tty 장치를 사용하지 않을 때 실패하는 한 테스트를 삭제하고 다른 두 가지를 수정한다. 테스트 스위트를 실행하려면 다음 명령을 실행하라:

```
sed -i -r 's|(pmap_initname)₩₩₩$|₩1|' testsuite/pmap.test/pmap.exp
sed -i '/set tty/d' testsuite/pkill.test/pkill.exp
rm testsuite/pgrep.test/pgrep.exp
make check
```

패키지를 설치한다:

```
make install
```

마지막으로, 필수 라이브러리들을 /usr가 마운트되지 않았을 때에도 찾을 수 있는 위치로 이동한다.

```
mv -v /usr/lib/libprocps.so.* /lib
ln -sfv ../../lib/$(readlink /usr/lib/libprocps.so) /usr/lib/libprocps.so
```

6.73.2. Procps-ng의 내용

설치된 프로그램들: free, pgrep, pidof, pkill, pmap, ps, pwdx, slabtop, sysctl, tload, top, uptime, vmstat, w, watch
설치된 라이브러리: libprocps.so
설치된 디렉토리들: /usr/include/proc와 /usr/share/doc/procps-ng-3.3.15

요약

free 시스템에서 사용 가능한 메모리 및 사용된 메모리 양(물리적 메모리 및 스왑 메모리 모두) 출력

pgrep	이름과 다른 속성들을 기준으로 프로세스 검색
pidof	지정된 프로그램의 PID 출력
pkill	이름 및 기타 속성에 따라 프로세스 신호 전송
pmap	지정된 프로세스의 메모리 맵 출력
ps	현재 실행 중인 프로세스 나열
pwdx	프로세스의 현재 작업 디렉토리 출력
slabtop	상세 커널 슬랩(slab) 캐시 정보를 실시간으로 표시
sysctl	런타임에 커널 매개 변수 수정
tload	현재 시스템 부하 평균 그래프 출력
top	가장 CPU 점유율이 높은 프로세스들의 목록을 출력; 프로세서 활동을 실시간으로 볼 수 있음
uptime	시스템 작동 시간, 로그온한 사용자 수 및 시스템 부하 평균 출력
vmstat	프로세스, 메모리, 페이징, 블록 입/출력 (IO), 트랩 및 CPU 활동에 대한 정보를 제공하는 가상 메모리 통계 출력
w	현재 로그온한 사용자와 로그온한 장치, 로그온 경과 시간들을 표시
watch	지정된 명령을 반복적으로 실행하여 출력이 가득 찬 첫 번째 화면을 표시; 이를 통해 사용자는 시간 경과에 따른 결과 출력의 변화를 볼 수 있음
libprocps	이 패키지의 대부분의 프로그램에서 사용하는 함수 포함

6.74. Util-linux-2.35.1

Util-linux 패키지는 다양한 유틸리티 프로그램을 포함하고 있다. 그 중에는 파일 시스템, 콘솔, 파티션 및 메시지를 처리하는 유틸리티들이 있다.

예상 빌드 시간: 1.1 SBU
필요 디스크 공간: 289 MB

6.74.1. FHS 준수 사항

FHS는 adjtime 파일을 일반적인 /etc 디렉토리 대신 /var/lib/hwclock 디렉토리에 둘 것을 권장한다. 먼저 hwclock 프로그램이 저장될 디렉토리를 생성한다:

```
mkdir -pv /var/lib/hwclock
```

6.74.2. Util-linux 설치

Util-linux 컴파일을 준비한다:

```
./configure ADJTIME_PATH=/var/lib/hwclock/adjtime   ₩
            --docdir=/usr/share/doc/util-linux-2.35.1 ₩
            --disable-chfn-chsh  ₩
            --disable-login      ₩
            --disable-nologin    ₩
            --disable-su         ₩
            --disable-setpriv    ₩
            --disable-runuser    ₩
            --disable-pylibmount ₩
            --disable-static     ₩
            --without-python     ₩
            --without-systemd    ₩
            --without-systemdsystemunitdir
```

--disable과 --without 옵션들은 LFS에 없는 패키지가 필요하거나 다른 패키지에 의해 설치된 프로그램과 호환되지 않는 구성 요소의 빌드에 대한 경고를 무시한다.

패키지를 컴파일한다:

```
make
```

테스트를 원한다면 root가 아닌 유저로 실행하라:

주의

테스트 스위트를 root 유저로 실행하는 것은 시스템에 피해를 끼칠 수 있다. 이를 실행하려면, 커널에 대한 CONFIG_SCSI_DEBUG 옵션을 반드시 실행 중인 시스템에서 사용할 수 있어야 하며, 반드시 모듈로 빌드되어야 한다. 커널에 포함시켜서 빌드하면 부팅이 되지 않는다. 완전한 테스트를 위해선 다른 BLFS 패키지를 설치해야 한다. 원한다면 이 테스트는 완성된 LFS 시스템으로 재부팅해서 다음 명령으로 실행할 수 있다:

```
bash tests/run.sh --srcdir=$PWD --builddir=$PWD
```

```
chown -Rv nobody .
su nobody -s /bin/bash -c "PATH=$PATH make -k check"
```

패키지를 설치한다:

```
make install
```

6.74.3. Util-linux의 내용

설치된 프로그램들:	addpart, agetty, blkdiscard, blkid, blkzone, blockdev, cal, cfdisk, chcpu, chmem, choom, chrt, col, colcrt, colrm, column, ctrlaltdel, delpart, dmesg, eject, fallocate, fdformat, fdisk, fincore, findfs, findmnt, flock, fsck, fsck.cramfs, fsck.minix, fsfreeze, fstrim, getopt, hexdump, hwclock, i386, ionice, ipcmk, ipcrm, ipcs, isosize, kill, last, lastb (last로 링크), ldattach, linux32, linux64, logger, look, losetup, lsblk, lscpu, lsipc, lslocks, lslogins, lsmem, lsns, mcookie, mesg, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, mountpoint, namei, nsenter, partx, pivot_root, prlimit, raw, readprofile, rename, renice, resizepart, rev, rfkill, rtcwake, script, scriptreplay, setarch, setsid, setterm, sfdisk, sulogin, swaplabel, swapoff (swapon로 링크), swapon, switch_root, taskset, ul, umount, uname26, unshare, utmpdump, uuidd, uuidgen, uuidparse, wall, wdctl, whereis, wipefs, x86_64, zramctl
설치된 라이브러리들:	libblkid.so, libfdisk.so, libmount.so, libsmartcols.so, libuuid.so
설치된 디렉토리들:	/usr/include/blkid, /usr/include/libfdisk, /usr/include/libmount, /usr/include/libsmartcols, /usr/include/uuid, /usr/share/doc/util-linux-2.35.1, /var/lib/hwclock

요약

addpart	리눅스 커널에 새 파티션 알림
agetty	tty 포트를 열고, 로그인 이름을 입력하라는 메시지를 표시한 다음, login 프로그램을 호출
blkdiscard	장치의 섹터 무시
blkid	블록 장치 속성을 찾아 출력하는 명령줄 유틸리티
blkzone	지정된 블록 장치에서 zone 명령 실행
blockdev	명령줄에서 블록 장치 ioctl을 호출
cal	간단한 달력을 표시
cfdisk	지정된 장치의 파티션 테이블 조작

chcpu	CPU 상태 수정
chmem	메모리 설정
choom	OOM-killer 점수를 표시하고 조정
chrt	프로세스의 실시간 속성 조작
col	Reverse line feeds를 필터링함
colcrt	오버 스트라이킹(overstriking) 및 하프 라인(half-lines)과 같은 일부 기능이 없는 터미널에 대한 nroff 출력 필터링
colrm	지정된 열 필터링
column	지정된 파일을 여러 열로 서식화
ctrlaltdel	Ctrl+Alt+Del 키 조합의 기능을 하드 또는 소프트 리셋으로 설정
delpart	리눅스 커널에서 파티션을 제거하도록 요청
dmesg	커널 부트 메시지 덤프
eject	이동식 미디어 추출
fallocate	파일의 공간을 사전 할당
fdformat	플로피 디스크를 로우 레벨로 포맷
fdisk	지정된 장치의 파티션 테이블 조작
fincore	코어에 있는 파일 내용의 페이지 수를 계산
findfs	레이블이나 Universally Unique Identifier (UUID)를 사용하여 파일 시스템 검색
findmnt	mountinfo, fstab 및 mtab 파일 작업을 위한 libmount 라이브러리에 대한 명령줄 인터페이스
flock	파일을 잠근 다음 잠금을 유지한 상태에서 명령 실행
fsck	파일 시스템을 확인하고 복구하는 데 사용
fsck.cramfs	지정된 장치의 Cramfs 파일 시스템에서 일관성 검사 수행
fsck.minix	지정된 장치의 Minix 파일 시스템에서 일관성 검사 수행
fsfreeze	FIFREEZE/FITHAW ioctl 커널 드라이버 작업을 다루는 매우 간단한 래퍼
fstrim	마운트된 파일 시스템에서 사용되지 않는 블록 삭제
getopt	지정된 명령줄의 옵션 구문 파싱
hexdump	지정된 파일을 16진수 또는 다른 지정된 형식으로 덤프
hwclock	Real-Time Clock (RTC) 또는 Basic Input-Output System (BIOS) 클럭이라고도 하는 시스템의 하드웨어 시계를 읽거나 설정 clock
i386	setarch로의 심볼릭 링크
ionice	프로그램에 대한 io 스케줄링 클래스 및 우선 순위를 가져오거나 설정
ipcmk	다양한 IPC 리소스 생성
ipcrm	지정된 프로세스 간 통신(Inter-Process Communication, IPC) 리소스 제거
ipcs	IPC 상태 정보 제공
isosize	iso9660 파일 시스템의 크기를 출력
kill	프로세스에 신호 전송

last	/var/log/wtmp 파일을 검색하고 마지막으로 로그인(및 로그아웃)한 유저를 표시; 시스템 부팅, 종료 및 run-level 변경 사항도 표시
lastb	/var/log/btmp 로그인 시 실패한 로그인 시도 표시
ldattach	직렬 장치 파일에 송/수신 데이터를 처리하기 위한 회선을 첨부
linux32	setarch로의 심볼릭 링크
linux64	setarch로의 심볼릭 링크
logger	시스템 로그에 지정된 메시지 입력
look	지정된 문자열로 시작하는 행 표시
losetup	루프 장치 설정 및 제어
lsblk	모든 또는 선택한 블록 장치에 대한 정보를 트리 형식으로 나열
lscpu	CPU 아키텍처 정보 출력
lsipc	시스템에서 현재 사용 중인 IPC 설비에 대한 정보 출력
lslocks	로컬 시스템 잠금(locks) 나열
lslogins	유저, 그룹 및 시스템 계정에 대한 정보 나열
lsmem	가용한 메모리의 범위와 작동 상태를 나열
lsns	네임스페이스 나열
mcookie	xauth용 매직 쿠키 생성 (128-bit 랜덤 16진수)
mesg	다른 사용자가 현재 사용자의 터미널로 메시지를 보낼 수 있는지의 여부 제어
mkfs	장치에 파일 시스템 구축(대개 하드 디스크 파티션)
mkfs.bfs	Santa Cruz Operations (SCO) bfs 파일 시스템 생성
mkfs.cramfs	Cramfs 파일 시스템 생성
mkfs.minix	Minix 파일 시스템 생성
mkswap	스왑 영역으로 사용될 지정된 장치나 파일을 초기화
more	한 번에 한 화면씩 텍스트를 페이징하는 필터
mount	지정된 장치의 파일 시스템을 파일 시스템 트리의 지정된 디렉토리에 연결
mountpoint	디렉토리가 마운트 지점인지 확인
namei	지정된 경로 이름의 심볼릭 링크를 표시
nsenter	다른 프로세스의 네임스페이스로 프로그램 실행
partx	커널에 디스크 파티션의 존재와 순번을 인식시킴
pivot_root	지정된 파일 시스템을 현재 프로세스의 새 root 파일 시스템으로 만듬
prlimit	프로세스의 리소스 제한 설정
raw	리눅스 raw 장치를 블록 장치에 바인딩
readprofile	커널 프로파일링 정보를 읽음
rename	지정된 파일의 이름을 변경하고 지정된 문자열을 다른 문자열로 대체
renice	프로세스 실행 우선순위 변경
resizepart	리눅스 커널에 파티션 크기 조정 요청

rev	지정된 파일의 행 반전
rkfill	무선 장치 활성화 및 비활성화 도구
rtcwake	지정된 절전 해제 시간까지 시스템 절전 상태로 들어가는 데 사용
script	터미널 세션의 타입스크립트 생성
scriptreplay	타이밍 정보를 사용해 타입스크립트 재생
setarch	새로운 프로그램 환경에서 보고된 아키텍쳐 변경 및 특성 플래그 설정
setsid	지정된 프로그램을 새 세션에서 실행
setterm	터미널 특성 설정
sfdisk	디스크 파티션 테이블 설정 도구
sulogin	root를 로그인할 수 있게 함; 일반적으로 시스템이 단일 사용자 모드로 전환될 때 init에 의해 호출됨
swaplabel	스왑 영역 UUID 및 레이블 변경
swapoff	페이징 및 스와핑을 위한 장치 및 파일 비활성화
swapon	페이징 및 스와핑을 위한 장치 및 파일 활성화, 현재 사용 중인 장치와 파일 나열
switch_root	마운트 트리의 root로 다른 파일 시스템으로 전환
tailf	로그 파일의 증가를 추적; 로그 파일의 마지막 10줄을 표시한 다음 로그 파일의 새 항목들을 계속 출력
taskset	프로세스의 CPU 선호도 검색 또는 설정
ul	밑줄을 터미널에서 밑줄을 나타내는 이스케이프 시퀀스로 변환하는 필터
umount	시스템의 파일 트리에서 파일 시스템 연결 해제
uname26	setarch로의 심볼릭 링크
unshare	상위 네임스페이스와 공유되지 않은 일부 네임스페이스로 프로그램 실행
utmpdump	지정된 로그인 파일의 내용을 보다 사용하기 쉬운 형식으로 표시
uuidd	UUID 라이브러리에서 안전하고 보장된 고유의 방식으로 시간 기반 UUID를 생성하기 위해 사용되는 데몬
uuidgen	새 UUID 생성. 각각의 새 UUID는 과거나 미래에 로컬 시스템과 다른 시스템에서 생성된 모든 UUID 중 고유한 값을 가진다.
uuidparse	고유 식별자를 파싱하는 유틸리티
wall	파일 내용이나, 기본적으로는 표준 입력을 현재 로그인한 모든 유저의 터미널에 출력
wdctl	하드웨어 watchdog 상태 표시
whereis	지정된 명령에 대한 바이너리, 출처, man 페이지의 위치를 출력
wipefs	장치에서 파일 시스템 서명 제거
x86_64	setarch로의 심볼릭 링크
zramctl	zram(압축된 램디스크) 장치를 설정하고 제어하는 프로그램
libblkid	장치 식별 및 토큰 추출 루틴 포함
libfdisk	파티션 테이블을 조작하는 루틴 포함
libmount	블록 장치 마운트 및 마운트 해제 루틴 포함

libsmartcols	표 형식의 화면 출력 지원 루틴 포함
libuuid	로컬 시스템 외부에서 액세스할 수 있는 개체의 고유 식별자를 생성하는 루틴 포함

6.75. E2fsprogs-1.45.5

E2fsprogs 패키지에는 ext2 파일 시스템을 처리하기 위한 유틸리티가 포함되어 있다. 또 ext3와 ext4 저널링 파일 시스템도 지원한다.

예상 빌드 시간: 1.6 SBU
필요 디스크 공간: 108 MB

6.75.1. E2fsprogs 설치

E2fsprogs 문서에서는 패키지를 소스 트리의 하위 디렉토리에 빌드할 것을 권장한다:

```
mkdir -v build
cd       build
```

E2fsprogs 컴파일을 준비한다:

```
../configure --prefix=/usr           ₩
             --bindir=/bin           ₩
             --with-root-prefix=""   ₩
             --enable-elf-shlibs     ₩
             --disable-libblkid      ₩
             --disable-libuuid       ₩
             --disable-uuidd         ₩
             --disable-fsck
```

환경 변수와 configure 옵션들의 의미:

--with-root-prefix=""와 --bindir=/bin
어떤 프로그램(예를 들면 e2fsck 프로그램)들은 필수적인 프로그램이다. 예를 들어 /usr가 마운트되지 않았을 때에도, 이러한 프로그램들은 사용 가능해야 한다. 이들은 /lib 및 /sbin과 같은 디렉토리에 속한다. 이 옵션이 E2fsprogs의 configure에 전달되지 않으면, 프로그램이 /usr 디렉토리에 설치된다.

--enable-elf-shlibs
이 패키지의 일부 프로그램이 사용하는 공유 라이브러리를 생성한다.

--disable-*
Util-Linux가 보다 최신 버전을 설치함에 따라 E2fsprogs가 libuuid 및 libblkid 라이브러리, uuidd 데몬, fsck 래퍼를 빌드하고 설치하는 것을 방지한다.

패키지를 컴파일한다:

```
make
```

결과를 테스트하려면 실행하라:

```
make check
```

E2fsprogs 테스트 중 하나는 256 MB의 메모리를 할당하려고 할 것이다. 이보다 훨씬 더 많은 RAM이 없는 경우 테스트를 위해 충분한 스왑 공간을 사용하도록 설정하라. 스왑 공간 생성 및 활성화에 대한 자세한 내용은 2.5절. “파티션에 파일 시스템 만들기” 및 2.7절. “새 파티션 마운팅”를 참고하라.

패키지를 설치한다:

```
make install
```

디버깅 심볼을 나중에 제거할 수 있도록 설치된 정적 라이브러리를 쓰기 가능하게끔 설정하라:

```
chmod -v u+w /usr/lib/{libcom_err,libe2p,libext2fs,libss}.a
```

이 패키지는 gzip으로 압축된 .info 파일을 설치하지만 시스템 전체 dir 파일을 업데이트하지는 않는다. 다음 명령을 사용하여 이 파일의 압축을 푼 후 시스템 dir 파일을 업데이트하라:

```
gunzip -v /usr/share/info/libext2fs.info.gz
install-info --dir-file=/usr/share/info/dir /usr/share/info/libext2fs.info
```

필요하다면, 다음 명령으로 추가적인 문서 몇 가지를 생성하고 설치하라:

```
makeinfo -o      doc/com_err.info ../lib/et/com_err.texinfo
install -v -m644 doc/com_err.info /usr/share/info
install-info --dir-file=/usr/share/info/dir /usr/share/info/com_err.info
```

6.75.2. E2fsprogs의 내용

설치된 프로그램들:	badblocks, chattr, compile_et, debugfs, dumpe2fs,e2freefrag, e2fsck, e2image, e2label, e2mmpstatus, e2scrub, e2scrub_all, e2undo, e4crypt, e4defrag, filefrag, fsck.ext2, fsck.ext3, fsck.ext4, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mkfs.ext4, mklost+found, resize2fs, tune2fs
설치된 라이브러리들:	libcom_err.so, libe2p.so, libext2fs.so, libss.so
설치된 디렉토리들:	/usr/include/e2p, /usr/include/et, /usr/include/ext2fs, /usr/include/ss, /usr/lib/e2fsprogs, /usr/share/et, /usr/share/ss

요약

badblocks	장치(보통 디스크 파티션)에서 잘못된 블록 검색
chattr	ext2 파일 시스템에서 파일 속성 변경; ext2 파일 시스템의 저널링 버전인 ext3 파일 시스템 변경
compile_et	에러 테이블 컴파일러; 에러 코드 이름 및 메시지 테이블을 com_err 라이브러리와 함께 사용하기에 적합한 C 소스 파일로 변환
debugfs	파일 시스템 디버거; ext2 파일 시스템의 상태를 검사하고 변경하는 데 사용할 수 있음
dumpe2fs	지정된 장치에 있는 파일 시스템에 대한 수퍼 블록 및 블록 그룹 정보 출력
e2freefrag	여유 공간 조각화 정보를 출력
e2fsck	ext2 파일 시스템과 ext3 파일 시스템을 확인하고 선택적으로 복구 하는데 사용
e2image	중요한 ext2 파일 시스템 데이터를 파일에 저장하는 데 사용
e2label	지정된 장치에 있는 ext2 파일 시스템의 레이블 표시 또는 변경
e2mmpstatus	ext4 파일 시스템의 MMP 상태 확인
e2scrub	마운트된 ext[234] 파일 시스템의 내용 확인
e2scrub_all	마운트된 모든 ext[234] 파일 시스템에 오류가 있는지 확인
e2undo	장치에 있는 ext2/ext3/ext4 파일 시스템에 대해 실행 취소 로그인 undo_log를 재생 [이는 e2fsprogs 프로그램에서 실패한 작업을 실행 취소하는 데 사용할 수 있다.]
e4crypt	Ext4 파일 시스템 암호화 유틸리티
e4defrag	ext4 파일 시스템용 조각 모음 도구

filefrag	특정 파일이 얼마나 심하게 파편화되었는지 출력
fsck.ext2	기본적으로 ext2 파일 시스템을 검사하는 e2fsck에 대한 하드 링크
fsck.ext3	기본적으로 ext3 파일 시스템을 검사하는 e2fsck에 대한 하드 링크
fsck.ext4	기본적으로 ext4 파일 시스템을 검사하는 e2fsck에 대한 하드 링크
logsave	명령의 출력을 로그 파일에 저장
lsattr	두 번째 확장 파일 시스템에 있는 파일 속성 나열
mk_cmds	명령 이름 및 도움말 메시지의 테이블을 libss 하위 시스템 라이브러리와 함께 사용하기 적합한 C 소스 파일로 변환
mke2fs	지정된 장치에 ext2나 ext3 파일 시스템 생성
mkfs.ext2	기본적으로 ext2 파일 시스템을 생성하는 mke2fs에 대한 하드 링크
mkfs.ext3	기본적으로 ext3 파일 시스템을 생성하는 mke2fs에 대한 하드 링크
mkfs.ext4	기본적으로 ext4 파일 시스템을 생성하는 mke2fs에 대한 하드 링크
mklost+found	ext2 파일 시스템에 lost+found 디렉토리를 생성하는 데 사용; e2fsck의 작업을 가볍게 하기 위해 디스크 블록을 이 디렉토리에 사전 할당함
resize2fs	ext2 파일 시스템을 확장하거나 축소하는 데 사용
tune2fs	ext2 파일 시스템에서 조정 가능한 파일 시스템 매개 변수 조정
libcom_err	일반 오류 표시 루틴
libe2p	dumpe2fs, chattr, lsattr에서 사용
libext2fs	유저-레벨 프로그램이 ext2 파일 시스템을 조작할 수 있게 하는 루틴 포함
libss	debugfs에서 사용

6.76. Sysklogd-1.5.1

Sysklogd 패키지에는 예기치 않은 상황이 발생했을 때 커널이 생성하는 것과 같은 시스템 메시지들을 기록하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 0.6 MB

6.76.1. Sysklogd 설치

우선 klogd가 특정 조건에서 세그멘테이션 오류를 일으키는 문제를 해결하고, 오래된 프로그램 구조를 수정한다:

```
sed -i '/Error loading kernel symbols/{n;n;d}' ksym_mod.c
sed -i 's/union wait/int/' syslogd.c
```

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make BINDIR=/sbin install
```

6.76.2. Sysklogd 설정

다음을 실행하여 새 /etc/syslog.conf 파일을 생성하라:

```
cat > /etc/syslog.conf << "EOF"
# Begin /etc/syslog.conf

auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *

# End /etc/syslog.conf
EOF
```

6.76.3. Sysklogd의 내용

설치된 프로그램들: klogd와 syslogd

요약

klogd 커널 메시지 인터셉트 및 로깅을 위한 시스템 데몬

syslogd 시스템 프로그램이 기록을 위해 제공하는 메시지 로깅[모든 로그 메시지에는 기본적으로 날짜와 호스트 이름이 포함되며, 일반적으로 프로그램 이름도 포함되지만, 로그 데몬을 얼마나 신뢰하는지에 따라 달라진다.]

6.77. Sysvinit-2.96

Sysvinit 패키지에는 시스템의 시작, 실행 및 종료를 제어하는 프로그램이 포함되어 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 1.4 MB

6.77.1. Sysvinit 설치

먼저 다른 패키지에 의해 설치된 여러 프로그램을 제거하고, 메시지를 명확히 하고 컴파일러 경고를 수정하는 패치를 적용한다:

```
patch -Np1 -i ../sysvinit-2.96-consolidated-1.patch
```

패키지를 컴파일한다:

```
make
```

이 패키지에는 테스트 스위트가 포함되어 있지 않다.

패키지를 설치한다:

```
make install
```

6.77.2. Sysvinit의 내용

설치된 프로그램들: bootlogd, fstab-decode, halt, init, killall5, poweroff (halt로 링크), reboot (halt로 링크), runlevel, shutdown, telinit (init로 링크)

요약

bootlogd	부팅 메시지를 로그 파일에 기록
fstab-decode	fstab-인코딩 인수를 사용하여 명령 실행
halt	보통은 -h 옵션으로 shutdown을 호출하지만, 이미 run-level이 0이라면 커널에 시스템을 중지하도록 지시한다; /var/log/wtmp 파일에 시스템이 종료되고 있음을 기록함
init	커널이 부팅 프로세스를 거치는 하드웨어들을 초기화할 때 실행하는 첫 번째 프로세스이자, 구성 파일에 지정된 모든 프로세스를 시작하는 프로세스
killall5	부모 셸을 죽이지 않도록 자체 세션의 프로세스를 제외한 모든 프로세스에 신호 전송
poweroff	커널에 시스템을 중지하고 컴퓨터를 끄도록 지시(halt 참고)
reboot	커널에 시스템 재부팅 지시 (halt 참고)
runlevel	/var/run/utmp의 마지막 run-level 기록에 적힌 이전 및 현재 run-level 출력
shutdown	모든 프로세스에 신호를 보내고 로그인한 모든 유저에게 통지해서 안전하게 시스템 종료
telinit	init에게 어떤 run-level로 변경할지를 지정

6.78. Eudev-3.2.9

Eudev 패키지에는 장치 노드의 동적 생성을 위한 프로그램이 포함되어 있다.

예상 빌드 시간: 0.2 SBU
필요 디스크 공간: 83 MB

6.78.1. Eudev 설치

Eudev 컴파일을 준비한다:

```
./configure --prefix=/usr           ₩
            --bindir=/sbin          ₩
            --sbindir=/sbin         ₩
            --libdir=/usr/lib       ₩
            --sysconfdir=/etc       ₩
            --libexecdir=/lib       ₩
            --with-rootprefix=      ₩
            --with-rootlibdir=/lib  ₩
            --enable-manpages       ₩
            --disable-static
```

패키지를 컴파일한다:

```
make
```

테스트와 설치 과정 모두에 쓰일 디렉토리들을 생성한다:

```
mkdir -pv /lib/udev/rules.d
mkdir -pv /etc/udev/rules.d
```

결과를 테스트하려면 실행하라:

```
make check
```

패키지를 설치한다:

```
make install
```

LFS 환경에서 유용한 몇 가지 사용자 지정 규칙 및 지원 파일 설치:

```
tar -xvf ../udev-lfs-20171102.tar.xz
make -f udev-lfs-20171102/Makefile.lfs install
```

6.78.2. Eudev 설정

하드웨어 장치에 대한 정보는 /etc/udev/hwdb.d 및 /lib/udev/hwdb.d 디렉토리에서 관리된다. Eudev는 그 정보를 바이너리 데이터베이스 /etc/udev/hwdb.bin으로 컴파일해야 한다. 초기 데이터베이스를 생성하라:

```
udevadm hwdb --update
```

하드웨어 정보가 업데이트될 때마다 이 명령을 실행해야 한다.

6.78.3. Eudev의 내용

설치된 프로그램들: udevadm, udevd
설치된 라이브러리들: libudev.so
설치된 디렉토리들: /etc/udev, /lib/udev, /usr/share/doc/udev-udev-lfs-20171102

요약

udevadm	일반 udev 관리 도구: udevd 데몬 제어, Udevd 데이터베이스에서 정보 제공, uevents 모니터링, uevents 완료 대기, Udev 구성 테스트 및 특정 장치에 대한 uevents 발동
udevd	넷링크 소켓에서 uevents를 수신하고 장치를 생성하며 이러한 uevents에 대응하여 구성된 외부 프로그램을 실행하는 데몬
libudev	udev 장치 정보에 대한 라이브러리 인터페이스
/etc/udev	Udev 구성 파일, 장치 권한 및 장치 명명 규칙 포함

6.79. 디버깅 심볼에 관하여

기본적으로 대부분의 프로그램과 라이브러리들은 디버깅 심볼이 포함되어(gcc의 -g 옵션과 함께) 컴파일된다. 이는 디버깅 정보가 포함된 프로그램이나 라이브러리를 디버깅할 때, 디버거가 메모리 주소뿐만 아니라 루틴과 변수의 이름들까지도 제공할 수 있다는 뜻이다.

그러나, 이런 디버깅 심볼을 포함하면 프로그램이나 라이브러리가 상당히 커지게 된다. 다음은 이러한 심볼들이 차지하는 공간에 대한 예시이다:

- 디버깅 심볼을 포함한 bash 바이너리: 1200 KB
- 디버깅 심볼을 제외한 bash 바이너리: 480 KB
- 디버깅 심볼을 포함한 Glibc와 GCC 파일들(/lib 및 /usr/lib): 87 MB
- 디버깅 심볼을 제외한 Glibc와 GCC 파일들: 16 MB

어떤 컴파일러와 C 라이브러리를 사용했느냐에 따라 크기가 달라질 수 있지만, 디버깅 심볼이 있는 프로그램과 없는 프로그램을 비교할 때는 보통 2 ~ 5배가량 차이가 난다.

대부분의 사용자들은 그들의 시스템 소프트웨어에서 디버거를 사용할 일이 전혀 없을 것이기 때문에, 이러한 심볼을 제거함으로써 많은 디스크 공간을 되찾을 수 있다. 다음 절에서는 프로그램과 라이브러리에서 모든 디버깅 심볼을 제거하는 방법을 설명한다.

6.80. 한 번 더 스트리핑하기

이 절은 선택 사항이다. 사용자가 프로그래머가 아니고 시스템 소프트웨어에서 디버깅을 할 계획이 없다면, 바이너리와 라이브러리에서 디버깅 심볼을 제거해서 시스템 크기를 약 90 MB 줄일 수 있다. 이는 소프트웨어를 더 이상 온전히 디버깅할 수 없다는 점 외에는 다른 불편을 일으키지 않는다.

아래에 언급된 명령어를 사용하는 대부분의 사람들은 어떠한 어려움도 겪지 않는다. 그러나, 오타를 쳐서 새 시스템이 사용할 수 없게 돼버리기 쉬우므로, strip 명령을 실행하기 전에 현재 상태에서 LFS 시스템을 백업하는 것이 좋다.

먼저 선택한 라이브러리의 디버깅 심볼을 별도의 파일에 배치한다. 이 디버깅 정보는 나중에 BLFS에서 valgrind나 gdb를 사용하는 회귀 테스트를 실행할 때 필요하다.

```
save_lib="ld-2.31.so libc-2.31.so libpthread-2.31.so libthread_db-1.0.so"

cd /lib

for LIB in $save_lib; do
    objcopy --only-keep-debug $LIB $LIB.dbg
    strip --strip-unneeded $LIB
    objcopy --add-gnu-debuglink=$LIB.dbg $LIB
done

save_usrlib="libquadmath.so.0.0.0 libstdc++.so.6.0.27
             libitm.so.1.0.0 libatomic.so.1.2.0"

cd /usr/lib

for LIB in $save_usrlib; do
    objcopy --only-keep-debug $LIB $LIB.dbg
    strip --strip-unneeded $LIB
    objcopy --add-gnu-debuglink=$LIB.dbg $LIB
done

unset LIB save_lib save_usrlib
```

스트리핑을 수행하기 전에, 스트리핑하려는 바이너리가 실행되지 않도록 각별히 주의하라:

```
exec /tools/bin/bash
```

이제 바이너리와 라이브러리는 안전하게 스트리핑될 수 있다:

```
/tools/bin/find /usr/lib -type f -name ₩*.a ₩
   -exec /tools/bin/strip --strip-debug {} ';'

/tools/bin/find /lib /usr/lib -type f ₩( -name ₩*.so* -a ! -name ₩*dbg ₩) ₩
   -exec /tools/bin/strip --strip-unneeded {} ';'

/tools/bin/find /{bin,sbin} /usr/{bin,sbin,libexec} -type f ₩
    -exec /tools/bin/strip --strip-all {} ';'
```

많은 파일로부터 파일 형식을 인식하지 못했다고 보고받을 것이다. 이러한 경고는 무시해도 무방하다. 이는 해당 파일이 바이너리 파일이 아닌 스크립트라는 것을 의미한다.

6.81. 정리하기

마지막으로, 테스트를 실행하고 남은 일부 추가 파일들을 정리한다:

```
rm -rf /tmp/*
```

이제 로그아웃한 후 업데이트된 chroot 명령을 사용하여 chroot 환경으로 다시 진입하라. 이제부터는, 종료 후에 chroot 환경으로 다시 진입해야 할 때 이 업데이트된 chroot 명령을 사용하라:

```
logout

chroot "$LFS" /usr/bin/env -i          ₩
   HOME=/root TERM="$TERM"             ₩
   PS1='(lfs chroot) ₩u:₩w₩$ '         ₩
   PATH=/bin:/usr/bin:/sbin:/usr/sbin ₩
   /bin/bash --login
```

이렇게 하는 이유는 /tools의 프로그램들이 더 이상 필요하지 않기 때문이다. 따라서 원한다면 /tools 디렉토리를 삭제해도 좋다.

참고

/tools를 삭제하면 툴체인 테스트 실행에 사용되었던 Tcl, Expect, DejaGNU의 임시 복사본도 삭제된다. 나중에 이러한 프로그램이 필요하면 다시 컴파일하고 다시 설치해야 한다. BLFS 책에는 이에 대한 지침이 수록되어 있다(http://www.linuxfromscratch.org/blfs/ 참고).

재부팅으로 인해 가상 커널 파일 시스템이 마운트 해제되었거나 수동으로 해제했다면 chroot로 다시 진입할 때 가상 커널 파일 시스템이 마운트되었는지 확인하라. 이 과정은 앞서 6.2.2절. “/dev 마운팅과 설정” 및 6.2.3절. “가상 커널 파일 시스템 마운팅”에서 설명되었다.

여러 패키지의 회귀 테스트를 통과하기 위해 이 장 앞 부분에서 지우지 않은 정적 라이브러리들이 몇 개 있었다. 이 라이브러리들은 binutils, bzip2, e2fsprogs, flex, libtool, zlib에 있던 것들이다. 원한다면 지금 삭제하라:

```
rm -f /usr/lib/lib{bfd,opcodes}.a
rm -f /usr/lib/libbz2.a
rm -f /usr/lib/lib{com_err,e2p,ext2fs,ss}.a
rm -f /usr/lib/libltdl.a
rm -f /usr/lib/libfl.a
rm -f /usr/lib/libz.a
```

/usr/lib 및 /usr/libexec 디렉토리에도 파일 이름의 확장자가 .la인 여러 파일들이 설치되었다. 이는 "libtool 아카이브" 파일이며 일반적으로 리눅스 시스템에서는 필요하지 않다. 이 시점에서는 이 중 어느 것도 필요하지 않다. 제거하려면 다음을 실행하라:

```
find /usr/lib /usr/libexec -name ₩*.la -delete
```

libtool 아카이브 파일에 대한 자세한 내용은 BLFS 섹션 "Libtool 아카이브(.la) 파일에 대하여"를 참고하라.

7장. 시스템 설정

7.1. 도입

리눅스 시스템 부팅은 여러 작업들을 수반한다. 이 과정 중에는 반드시 가상, 실제 파일 시스템을 모두 마운트하고, 장치 초기화, 스왑 활성화, 파일 시스템 무결성 확인, 스왑 파티션 또는 파일 마운트, 시스템 시계 설정, 네트워크 시작, 시스템에 필요한 데몬 시작, 사용자가 필요로 하는 사용자 지정 작업들을 수행해야 한다. 이 절차는 작업들이 올바른 순서로 수행되도록 구성되어야 하지만 동시에 가능한 한 빨리 실행되어야 한다.

7.1.1. System V

System V는 약 1983년부터 리눅스 등 유닉스와 유닉스 계열 시스템에서 사용되고 있는 고전적인 부팅 절차이다. login(getty를 통한)같은 기본 프로그램을 시작하고 스크립트를 실행하는, init이라는 작은 프로그램으로 구성되어 있다. 보통 rc라는 이름의 이 스크립트는 시스템 초기화에 필요한 작업을 수행하는 추가적인 스크립트들의 실행을 관리한다.

init 프로그램은 /etc/inittab 파일에 의해 제어되며 사용자가 실행할 수 있는 run level들로 구성되어 있다:

```
0 — 시스템 종료(halt)
1 — 단일 사용자 모드(Single user mode)
2 — 네트워크 미지원 다중 사용자 모드(Multiuser, without networking)
3 — 다중 사용자 모드(Full multiuser mode)
4 — 사용자 정의 가능(User definable)
5 — 디스플레이 관리자가 있는 다중 사용자 모드(Full multiuser mode with display manager)
6 — 재부팅(reboot)
```

보통 기본 run level은 3이나 5이다.

장점

- 철저히 분석되어 확고하게 자리잡은 검증된 시스템.
- 쉽게 사용자 정의 가능.

단점

- 비교적 느린 부팅 속도. 평균적인 속도의 LFS 시스템은 첫 번째 커널 메시지부터 로그인 프롬프트까지 부팅 시간을 측정했을 때 8-12초가 걸린다. 네트워크 연결은 일반적으로 로그인 프롬프트 후 약 2초 후에 수립된다.
- 부팅 작업의 직렬 처리. 이것은 위에서 말한 바와 관련이 있다. 파일 시스템 확인과 같은 어떤 절차가 지연되면, 전체 부팅 절차가 지연된다.
- 제어 그룹(cgroups), 사용자별 공평 분배 스케줄링과 같은 고급 기능을 직접 지원하지 않는다.
- 스크립트를 추가하려면 수동적이고 고정적인 순서대로 판단해야 한다.

7.2. LFS-Bootscripts-20191031

LFS-Bootscripts 패키지에는 부팅/종료 시 LFS 시스템을 시작/중지하는 스크립트들이 포함되어 있다. 부팅 프로세스를 사용자 지정하는 데 필요한 구성 파일 및 절차는 다음 절에 설명되어 있다.

예상 빌드 시간: 0.1 SBU 이하
필요 디스크 공간: 244 KB

7.2.1. LFS-Bootscripts 설치

패키지를 설치한다:

```
make install
```

7.2.2. LFS-Bootscripts의 내용

설치된 스크립트들: checkfs, cleanfs, console, functions, halt, ifdown, ifup, localnet, modules, mountfs, mountvirtfs, network, rc, reboot, sendsignals, setclock, ipv4-static, swap, sysctl, sysklogd, template, udev, udev_retry
설치된 디렉토리들: /etc/rc.d, /etc/init.d (심볼릭 링크), /etc/sysconfig, /lib/services, /lib/lsb (심볼릭 링크)

요약

checkfs	파일 시스템을 마운트하기 전에 무결성 확인(저널 및 네트워크 기반 파일 시스템은 제외)
cleanfs	/var/run/과 /var/lock/의 파일들과 같이 재부팅할 때 보존되지 않아야 하는 파일들을 제거; /var/run/utmp를 다시 생성하고 존재할 수 있는 /etc/nologin, /fastboot, /forcefsck 파일들을 삭제함
console	원하는 키보드 레이아웃에 적합한 키맵 테이블을 로드; 화면 글꼴도 설정
functions	여러 부팅 스크립트에서 사용되는 오류 및 상태 확인과 같은 공통 기능 포함
halt	시스템 종료
ifdown	네트워크 장치 중지
ifup	네트워크 장치 초기화
localnet	시스템의 호스트 이름과 로컬 루프백 장치 설정
modules	/etc/sysconfig/modules에 나열된 커널 모듈들을, 그 파일에 지정된 인수들을 사용해서 로드
mountfs	noauto로 표시되거나 네트워크 기반인 파일 시스템을 제외한 모든 파일 시스템 마운트
mountvirtfs	proc와 같은 가상 커널 파일 시스템 마운트
network	네트워크 카드와 같은 네트워크 인터페이스 설정 및 기본 게이트웨이 설정(해당하는 경우)
rc	마스터 run-level 제어 스크립트; 처리 중인 심볼릭 링크의 이름에 따라 결정된 순서대로 다른 모든 부트 스크립트를 하나씩 실행하도록 되어 있음.
reboot	시스템 재시작
sendsignals	시스템이 재부팅 또는 중지되기 전에 모든 프로세스가 종료되었는지 확인
setclock	하드웨어 시계가 UTC 시간으로 설정되지 않은 경우 커널 시계를 로컬 시간으로 재설정
ipv4-static	정적 인터넷 프로토콜(Internet Protocol, IP) 주소를 네트워크 인터페이스에 할당하는 데 필요한 기능 제공

swap	스왑 파일 및 파티션 활성화 및 비활성화
sysctl	/etc/sysctl.conf 파일이 존재한다면, 그로부터 실행 중인 커널로 시스템 설정 값 로드
sysklogd	시스템과 커널 로그 데몬 시작 및 중지
template	다른 데몬에 대한 사용자 지정 부트스크립트를 만드는 템플릿
udev	/dev 디렉토리 준비 및 Udev 시작
udev_retry	실패한 udev uevents를 재시도하고, 필요하다면 생성된 규칙 파일을 /run/udev에서 /etc/udev/rules.d로 복사

7.3. 장치 및 모듈 핸들링 개요

6장에서 우리는, eudev가 빌드될 때 Udev 패키지를 설치했다. 이들의 작동 방식에 대한 자세한 내용을 살펴보기 전에, 과거의 장치 핸들링 방법의 역사에 대해 간단히 짚고 넘어가고자 한다.

일반적인 리눅스 시스템은 전통적으로 정적인 장치 생성 방법을 사용해서, 해당 하드웨어 장치가 실제로 존재하는지 여부에 관계없이 /dev 아래에 매우 많은 장치 노드(때로는 말 그대로 수천 개의 노드)가 만들어졌다. 이는 일반적으로 MAKEDEV 스크립트를 통해 이뤄졌는데, 이 스크립트는 세상에 존재할 수 있는 모든 장치에 대한 관련 주/부 장치 번호와 함께 mknod 프로그램에 대한 여러 호출들을 담고 있었다.

Udev를 사용하면 커널에 의해 탐지된 장치만 해당 장치 노드가 생성된다. 이런 장치 노드는 시스템이 부팅될 때마다 생성되기 때문에 devtmpfs 파일 시스템(완전히 시스템 메모리에 상주하는 가상 파일 시스템)에 저장된다. 장치 노드는 공간이 많이 필요하지 않기 때문에 사용되는 메모리는 무시할 수 있다.

7.3.1. 역사

2000년 2월, devfs라는 새로운 파일 시스템이 2.3.46 커널에 병합되어 2.4 안정 버전 커널에서 사용할 수 있게 되었다. 동적으로 장치를 만드는 이 방법은 비록 커널 소스에 존재는 했으나 핵심 커널 개발자들로부터 결코 지지를 받지 못했다.

devfs에서 채택된 접근 방식의 주요 문제는 장치 인식, 생성 및 명명 처리 방식이었다. 장치 노드 명명 문제인 후자가 아마도 가장 중요한 사안이었을 것이다. 장치 이름을 설정하는 것이 가능하다면, 장치 명명 정책은 특정 개발자가 아닌 시스템 관리자의 몫이어야 한다는 것이 보편적인 생각이다. 더구나 devfs 파일 시스템은 그 설계 구조상 커널에 대한 실질적인 수정 없이는 고칠 수 없는 교착 상태에 시달렸다. 이는 유지보수가 부실해지자 장기간 폐기된 것으로 표시되었고, 결국 2006년 6월 커널에서 제거되었다.

나중에 2.6 안정 버전으로 배포된 2.5 개발 버전 커널 트리가 개발되면서 sysfs라는 새로운 가상 파일 시스템이 생겼다. sysfs의 역할은 시스템 하드웨어 구성을 유저스페이스 프로세스(userspace processes)에서 볼 수 있도록 하는 것이다. 이 유저스페이스에서 볼 수 있도록 하는 표현 방식으로 인해, devfs에 대응하는 유저스페이스용 대체품 개발 가능성이 훨씬 더 높아졌다.

7.3.2. Udev 구현

7.3.2.1. Sysfs

sysfs 파일 시스템은 위에서 간략히 언급했다. 그렇다면 sysfs는 시스템에 어떤 장치가 존재하는지, 또 그 장치들에 어떤 장치 번호를 사용해야 하는지를 어떻게 알 수 있을까? 커널 내부로 컴파일된 드라이버는 커널에 의해 감지되는 대로 장치를 sysfs(내부적으론 devtmpfs)에 직접 등록한다. 모듈로 컴파일된 드라이버는 모듈이 로드될 때 등록된다. sysfs 파일 시스템이 마운트되고나면(/sys에), 드라이버가 sysfs에 등록한 데이터는 유저스페이스 프로세스와 udevd에서 사용 가능하여 장치 노드 수정을 포함한 처리에 쓰인다.

7.3.2.2. 장치 노드 생성

장치 파일은 커널에 의해 devtmpfs 파일 시스템으로 생성된다. 장치 노드를 등록하려는 모든 드라이버는 devtmpfs를 통해 (드라이버 코어에 의해) 작업을 수행한다. devtmpfs 인스턴스가 /dev에 마운트되면, 장치 노드가 고정된 이름과 권한, 소유자를 갖고 초기화되어 생성된다.

그리고나서, 커널은 uevent를 udevd로 보낸다. /etc/udev/rules.d, /lib/udev/rules.d, /run/udev/rules.d 디렉토리에 있는 파일들에 지정된 규칙에 따라, udevd는 장치 노드에 대한 추가 심볼릭 링크를 만들거나, 장치 노드 권한, 소유자 또는 그룹을 변경하거나, 해당 개체에 대한 내부 udevd 데이터베이스 항목(이름)을 수정한다.

이 세 디렉토리 안의 규칙들은 번호가 메겨지고 디렉토리들은 모두 병합된다. 만약 udevd가 생성 중인 장치에 대한 규칙을 찾지 못하면, devtmpfs가 처음 사용된 장치의 권한과 소유권을 남긴다.

7.3.2.3. 모듈 로딩

모듈로 컴파일된 장치 드라이버에는 별칭(alias)이 내장되어 있을 수 있다. 별칭은 modinfo 프로그램의 출력을 통해 확인할 수 있으며 보통은 모듈이 지원하는 장치의 버스 고유 식별자와 관련이 있다. 예를 들어, snd-fm801 드라이버는 벤더 ID가 0x1319이고 장치 ID가 0x0801인 PCI 장치를 지원하며, “pci:v00001319d00000801sv*sd*bc04sc01i*”라는 별칭을 갖고 있다. 대부분의 장치에서, 버스 드라이버는 sysfs를 통해 장치를 제어할 드라이버의 별칭을 내보낸다. 다시 말해, /sys/bus/pci/devices/0000:00:0d.0/modalias 파일은 문자열 “pci:v00001319d00000801sv00001319sd00001319bc04sc01i00”을 담고 있다. Udev와 함께 제공되는 기본 규칙은 udevd가 MODALIAS uevent 환경 변수의 내용(sysfs의 modalias 파일 내용과 같음)으로 /sbin/modprobe를 호출하여, 와일드카드 확장 후 별칭이 이 문자열과 일치하는 모든 모듈을 로드한다.

앞서 제시한 snd-fm801 예시에서는, 구식(그리고 원치 않는) forte 드라이버가 사용 가능하다면, 추가적으로 로드된다는 것을 의미한다. 원치 않는 드라이버 로드를 방지하려면 아래를 참고하라.

커널 자체도 네트워크 프로토콜, 파일 시스템 및 NLS 지원을 위한 모듈을 필요에 따라 로드할 수 있다.

7.3.2.4. 핫플러그/동적 장치 처리

USB MP3플레이어와 같은 장치를 컴퓨터에 연결하면, 커널은 장치가 연결된 것을 인식하고 uevent를 생성한다. 이 uevent는 앞서 설명한 바와 같이 udevd에 의해 처리된다.

7.3.3. 모듈 로딩 및 장치 생성 문제

장치 노드를 자동으로 생성할 때는 몇 가지 문제가 발생할 수 있다.

7.3.3.1. 커널 모듈이 자동으로 로드되지 않음

Udev는 모듈이 버스 고유 별칭을 갖고 버스 드라이버가 필요한 별칭을 sysfs로 적절하게 내보내는 경우에만 모듈을 로드할 것이다. 그렇지 않은 경우에는 다른 방법으로 모듈을 로드해야 한다. Linux-5.5.3에서는, Udev가 제대로 작성된 INPUT, IDE, PCI, USB, SCSI, SERIO, FireWire 장치용 드라이버들을 로드하는 것으로 알려져 있다.

필요한 장치 드라이버가 Udev에 필요한 사항을 갖고있는지 확인하려면, modinfo에 모듈 이름을 인수로 전달해 실행하라. 이제 /sys/bus 아래에서 장치 디렉토리를 찾고 modalias 파일이 있는지 확인하라.

modalias 파일이 sysfs에 존재하면, 드라이버는 장치를 지원하고 직접 통신할 수 있지만, 별칭이 없다면 드라이버의 버그이다. Udev의 도움 없이 드라이버를 로드하고 나중에 문제가 수정되길 기대하라.

/sys/bus 아래의 관련 디렉토리에 modalias 파일이 없다면, 이는 커널 개발자들이 아직 이 버스 유형에 modalias 지원을 추가하지 않았음을 의미한다. Linux-5.5.3에서는, ISA 버스가 이에 해당한다. 이 문제는 이후 커널 버전에서 수정되기를 기대하라.

Udev는 snd-pcm-oss와 같은 “wrapper” 드라이버나 loop와 같은 비하드웨어 드라이버를 로드하기 위한 것이 전혀 아니다.

7.3.3.2. 커널 모듈이 자동으로 로드되지 않으며, Udev도 커널 모듈을 로드하지 않음

“wrapper” 모듈이 일부 다른 모듈에서 제공하는 기능만 향상시킨다면(가령, snd-pcm-oss는 OSS 응용 프로그램에서 사운드 카드를 사용할 수 있게 하여 snd-pcm의 기능을 향상시킴), Udev가 래핑된 모듈을 로드한 후 래퍼를 로드하도록 modprobe를 구성하라. 이렇게 하려면 해당 /etc/modprobe.d/<filename>.conf 파일에 “softdep” 줄을 추가하라. 예를 들면:

```
softdep snd-pcm post: snd-pcm-oss
```

“softdep” 명령은 pre: 구문이나 pre:와 post:의 혼용도 허용한다는 점을 참고하라. “softdep” 문법 및 기능에 대한 자세한 내용은 modprobe.d(5) 매뉴얼 페이지를 참고하라.

문제의 모듈이 래퍼가 아니며 그 자체로 유용할 경우, 그 모듈을 시스템 부팅 중 로드하도록 modules 부트스크립트를 구성하라. 이렇게 하려면 모듈 이름을 /etc/sysconfig/modules 파일에 별도의 행으로 추가하라. 이것은 래퍼 모듈에도 적용되지만, 그것은 차선책이다.

7.3.3.3. Udev가 일부 원치 않는 모듈을 로드함

모듈을 빌드하지 않도록 하거나, 아래의 forte 모듈 예시처럼 /etc/modprobe.d/blacklist.conf 파일에 모듈을 블랙리스트로 추가하라:

```
blacklist forte
```

블랙리스트된 모듈도 modprobe 명령을 명시적으로 사용하여 수동으로 로드할 수 있다.

7.3.3.4. Udev가 장치를 잘못 생성하거나, 잘못된 심볼릭 링크를 생성함

이런 현상은 일반적으로 규칙이 예기치 않게 장치와 일치할 때 발생한다. 예를 들어, 형편없이 작성된 규칙은 SCSI 디스크(의도한 장치)와 해당 벤더의 SCSI 일반 장치(잘못된) 양쪽에 일치할 수 있다. udevadm info 명령의 도움을 받아 위반되는 규칙을 찾고 보다 구체적으로 수정하라.

7.3.3.5. Udev 규칙을 신뢰할 수 없음

이것은 앞서 설명한 문제의 또다른 증상일 수 있다. 그렇지 않고 규칙이 sysfs 속성을 사용한다면, 이후 커널에서 수정될 수 있는 커널 타이밍 문제일 수 있다. 당장은 사용된 sysfs 속성을 기다리는 규칙을 만들어 /etc/udev/rules.d/10-wait_for_sysfs.rules 파일에 추가(이 파일이 존재하지 않으면 생성하라)하면 된다. 이렇게 해서 도움이 되었다면 LFS 개발 리스트에 알려주기 바란다.

7.3.3.6. Udev가 장치를 생성하지 않음

아래 문단은 드라이버가 커널에 정적으로 구축되었거나 이미 모듈로 로드되어 있으며, Udev가 잘못 명명된 장치를 생성하지 않는 것을 이미 확인했다고 가정한다.

커널 드라이버가 데이터를 sysfs로 내보내지 않으면 Udev에는 장치 노드를 생성하는 데 필요한 정보가 없다. 이는 커널 트리 외부의 타사 드라이버에서 가장 흔하게 발생한다. /lib/udev/devices에 적절한 주/부 번호로 정적 장치 노드를 생성하라(커널 문서 내의 devices.txt 파일이나 타사 드라이버 공급 업체가 제공한 문서를 참고). 정적 장치 노드는 udev에 의해 /dev에 복사될 것이다.

7.3.3.7. 재부팅 후 장치 이름 지정 순서가 임의로 변경됨

이는 Udev가 설계상 uevents와 모듈 로드를 병렬로 처리하기 때문에, 순서를 예측할 수 없기 때문이다. 이것은 결코 “수정”되지 않을 것이다. 장치 이름들이 안정적일 것으로 커널에 기대해선 안 된다. 대신, Udev가 설치한 다양한 *_id 유틸리티의 출력이나 일련 번호와 같이, 장치의 안정적인 속성을 기반으로 안정적인 이름을 가진 심볼릭 링크를 만드는 규칙을 직접 생성하라. 예시는 7.4절. “장치 관리”와 7.5절. “일반 네트워크 구성”를 참고하라.

7.3.4. 유용한 읽을거리

다음 사이트들에서 추가적으로 유용한 문서들을 읽을 수 있다:

- devfs의 유저스페이스 구현 http://www.kroah.com/linux/talks/ols_2003_udev_paper/Reprint-Kroah-Hartman-OLS2003.pdf
- sysfs 파일 시스템 http://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

7.4. 장치 관리

7.4.1. 네트워크 장치

Udev는 기본적으로 펌웨어/BIOS 데이터 또는 버스, 슬롯, MAC 주소와 같은 물리적 특성에 따라 네트워크 장치를 명명한다. 이 명명 규칙의 목적은 네트워크 장치 이름이 장치가 인식된 시간에 근거하지 않고 일관성 있게 명명되도록 하는 것이다. 예를 들어 인텔과 리얼텍의 네트워크 카드가 두 개 있는 컴퓨터에서, 인텔 네트워크 카드 이름이 eht0, 리얼텍 카드는 eth1이 되었다면 재부팅 후에는 카드 번호가 서로 반대로 매겨지기도 한다.

새로운 명명 체계에서 일반적인 네트워크 장치 이름은 enp5s0 또는 wlp3s0과 같은 것이 될 것이다. 이 명명 규칙을 원하지 않는다면, 전통적인 명명 방식이나 사용자 정의 방식도 사용할 수 있다.

7.4.1.1. 커널 명령줄에서 영구 명명 비활성화

eth0, eth1과 같이 명명하는 기존 방식은 커널 명령줄에 net.ifnames=0을 추가하여 복원할 수 있다. 이는 동일한 유형의 이더넷 장치가 단 한 개만 존재하는 시스템에 가장 적합하다. 노트북은 흔히 eth0과 wlan0이라는 이름으로 이더넷 연결 방식을 여러 개 가지고 있으며 이 방법을 사용할 법한 대상이기도 하다. 명령행은 GRUB 구성 파일에서 전달된다. 8.4.4절. “GRUB 구성 파일 생성”를 참고하라.

7.4.1.2. 사용자 지정 Udev 규칙 생성

명명 방식은 사용자 지정 Udev 규칙을 생성함으로써 변경할 수 있다. 초기 규칙을 생성하는 스크립트가 포함되어 있다. 다음을 실행하여 이러한 규칙들을 생성하라:

```
bash /lib/udev/init-net-rules.sh
```

이제 /etc/udev/rules.d/70-persistent-net.rules 파일을 열어 어떤 네트워크 장치에 어떤 이름이 할당되었는지 확인하라:

```
cat /etc/udev/rules.d/70-persistent-net.rules
```

참고

MAC 주소가 네트워크 카드에 수동으로 할당되었거나, Qemu, Xen과 같은 가상 환경에서 MAC 주소를 할당받은 경우와 같은 일부 상황에서는 주소가 일관되게 할당되지 않았기 때문에 네트워크 규칙 파일이 생성되지 않았을 수 있다. 이런 경우엔 이 방법을 사용할 수 없다.

이 파일은 각 NIC에 대해 두 줄씩 이어지는 주석 블록으로 시작한다. 각 NIC의 첫 번째 줄은 드라이버를 찾을 수 있는 경우 괄호 안의 드라이버와 함께 해당 하드웨어 ID(PCI 카드를 예로 들면 PCI 공급 업체와 장치 ID)를 보여주는 주석이다. 하드웨어 ID나 드라이버 모두 인터페이스를 제공할 이름을 명명하는 데 사용되지 않는다; 이 정보는 참조용일 뿐이다. 두 번째 줄이 이 NIC와 일치하며 실제로 이름을 할당하는 Udev 규칙이다.

모든 Udev 규칙은 쉼표와 선택적 공백으로 구분된 여러 개의 키로 구성된다. 이 규칙의 키와 각각의 설명은 다음과 같다:

- SUBSYSTEM=="net" - 네트워크 카드가 아닌 장치들을 무시한다.
- ACTION=="add" - add가 아닌 uevent에 대해서는 이 규칙을 무시하도록 한다("remove" 및 "change" uevents도 발생하지만 네트워크 인터페이스의 이름을 바꿀 필요는 없다).
- DRIVERS=="?*" - Udev가 VLAN이나 브릿지 하위 인터페이스를 무시하도록 한다(이 하위 인터페이스들은 드라이버를 갖지 않기 때문). 이 하위 인터페이스들은 할당될 이름이 부모 장치와 충돌하기 때문에 건너뛴다.
- ATTR{address} - NIC의 MAC 주소이다.

- ATTR{type}=="1" - 여러 가상 인터페이스를 생성하는 특정 무선 드라이버의 경우 규칙이 주 인터페이스와만 짝을 이루도록 한다. VLAN과 브릿지 하위 인터페이스와 같은 이유로 보조 인터페이스는 생략된다: 그렇지 않으면 이름이 충돌할 수 있다.
- NAME - Udev가 이 인터페이스에 할당할 이름이다.

NAME의 값이 중요한 부분이다. 계속하기 전에 각 네트워크 카드에 할당된 이름을 정확히 숙지하고 아래 구성 파일을 만들 때 해당 NAME 값을 사용하라.

7.4.2. CD-ROM symlinks

추후 설치할지도 모르는 일부 소프트웨어(예: 다양한 미디어 플레이어)에는 CD-ROM이나 DVD-ROM을 가리키는 /dev/cdrom 및 /dev/dvd 심볼릭 링크가 필요하다. 또 이러한 심볼릭 링크들을 참조하도록 /etc/fstab에 추가하는 것이 편리할 것이다. Udev에는 각 장치의 기능에 따라 이러한 심볼릭 링크를 생성하기 위해 규칙 파일을 생성하는 스크립트가 함께 제공되지만, 스크립트를 사용하도록 하려면 두 가지 작동 모드 중 어떤 모드를 사용할지 결정해야 한다.

첫째는 스크립트가 “by-path” 모드(USB 및 FireWire 장치에서 기본적으로 사용됨)로 작동할 수 있고, 여기서는 스크립트가 생성하는 규칙이 CD나 DVD 장치의 물리적 경로에 따라 달라진다. 둘째는 “by-id” 모드(IDE와 SCSI 장치에서의 기본값)로 작동할 수 있으며, 여기서 생성되는 규칙은 CD나 DVD 장치 자체에 저장된 식별 문자열에 따라 달라진다. 경로는 Udev의 path_id 스크립트에 의해 결정되며, 식별 문자열은 보유하고 있는 장치 유형에 따라 ata_id나 scsi_id 프로그램에 의해 하드웨어로부터 인식된다.

각 접근법마다 장점이 있다; 어떤 방법을 사용할지는 어떤 종류의 장치가 변경될 수 있는지에 따라 달라질 것이다. 드라이브를 다른 IDE 포트나 다른 USB 커넥터로 옮기려는 때와 같이, 장치에 대한 물리적 경로(즉, 연결되는 포트 또는 슬롯)가 변경될 것이라면 “by-id” 모드를 사용하라. 반면에 장치 수명이 끝나가서 동일한 기능의 다른 장치로 교체하고 동일한 커넥터에 꽂는다면, 장치의 식별이 변경될 것이기 때문에 “by-path” 모드를 사용해야 한다.

두 경우 모두 일어날 것 같다면, 더 자주 발생할 것으로 예상되는 상황에 따라 모드를 선택하라.

중요

외부 장치(예: USB 연결 CD 드라이브)는 장치를 새 외부 포트에 연결할 때마다 물리적 경로가 변경되기 때문에 by-path 모드를 사용해선 안 된다. 모든 외부 연결식 장치들은, 물리적 경로로 인식하도록 Udev 규칙을 작성할 경우 이 문제가 발생할 것이다; 이러한 문제는 CD와 DVD 드라이브에만 국한되지 않는다.

Udev 스크립트가 사용할 값을 보려면 해당 CD-ROM에 대해, /sys 아래(예를 들면 /sys/block/hdd)에서 해당 디렉토리를 찾아 다음과 유사한 명령을 실행하라:

```
udevadm test /sys/block/hdd
```

다양한 *_id 프로그램의 출력이 나오는 행을 살펴보라. “by-id” 모드는 ID_SERIAL 값이 존재하고 비어있지 않으면 그 값을 사용하고, 그렇지 않으면 ID_MODEL과 ID_REVISION의 조합을 사용한다. “by-path” 모드는 ID_PATH 값을 사용할 것이다.

기본 모드가 사용자의 상황에 적합하지 않은 경우 /etc/udev/rules.d/83-cdrom-symlinks.rules 파일을 다음과 같이 수정할 수 있다(여기서 모드는 “by-id” 또는 “by-path” 중 하나임):

```
sed -i -e 's/"write_cd_rules"/"write_cd_rules mode"/' ₩
   /etc/udev/rules.d/83-cdrom-symlinks.rules
```

지금은 규칙 파일이나 심볼릭 링크를 생성할 필요가 없다는 점을 참고하라. 왜냐하면 LFS 시스템에 호스트의 /dev 디렉토리를 바인딩 마운트했으며 호스트에 심볼릭 링크가 존재한다고 가정하기 때문이다. LFS 시스템을 처음 부팅하면 규칙과 심볼릭 링크가 생성될 것이다.

그러나, 만약 CD-ROM 장치가 여러 개 있다면 장치가 예측 가능한 순서대로 검색되지 않기 때문에, 해당 시점에 생성된 심볼릭 링크가 호스트에서 가리키던 장치와는 다른 장치를 가리킬 수 있다. LFS 시스템을 처음 부팅할 때 할당되는 링크는 안정적이기 때문에, 동일한 장치를 가리키는 데에 양쪽 시스템 모두 심볼릭 링크가 필요한 경우에만 문제가 된다. 필요하다면 부팅 후 생성된 /etc/udev/rules.d/70-persistent-cd.rules 파일을 검사하고 편집하여 할당된 심볼릭 링크가 필요한 것과 일치하도록 하라.

7.4.3. 중복 장치 처리

7.3절. “장치 및 모듈 핸들링 개요”에서 설명했듯이, 동일한 기능의 장치끼리는 /dev에 나타나는 순서가 본질적으로 무작위이다. 가령 USB 웹 카메라와 TV 튜너가 있을 때 /dev/video0은 카메라를, /dev/video1은 튜너를 참조할 때가 있고, 재부팅 후에는 순서가 반대로 될 수도 있다. 이 문제는 사운드 카드와 네트워크 카드를 제외한 모든 종류의 하드웨어에 대해 사용자 지정 영구 심볼릭 링크를 위한 Udev 규칙을 생성하여 해결할 수 있다. 네트워크 카드의 경우는 7.5절. “일반 네트워크 구성”에서 별도로 다루며, 사운드 카드 구성은 BLFS에서 확인할 수 있다.

이 문제가 발생할 가능성이 높은 각 장치에 대해(현재 사용 중인 리눅스 배포판에서 문제가 없더라도), /sys/class나 /sys/block 아래에서 해당 디렉토리를 찾아보라. 비디오 장치는 아마 /sys/class/video4linux/videoX일 것이다. 장치를 고유하게 식별하는 속성을 파악하라(보통은 공급 업체와 제품 ID 그리고/또는 시리얼 넘버가 이에 속한다):

```
udevadm info -a -p /sys/class/video4linux/video0
```

그리고나서 심볼릭 링크를 생성하는 규칙을 작성한다. 예를 들어:

```
cat > /etc/udev/rules.d/83-duplicate_devs.rules << "EOF"

# Persistent symlinks for webcam and tuner
KERNEL=="video*", ATTRS{idProduct}=="1910", ATTRS{idVendor}=="0d81", ₩
   SYMLINK+="webcam"
KERNEL=="video*", ATTRS{device}=="0x036f", ATTRS{vendor}=="0x109e", ₩
   SYMLINK+="tvtuner"

EOF
```

그 결과, /dev/video0과 /dev/video1 장치는 여전히 튜너와 웹 카메라를 무작위로 가리키겠지만(따라서 직접 사용해서는 안 된다), /dev/tvtuner와 /dev/webcam 심볼릭 링크는 항상 올바른 장치를 가리킬 것이다.

7.5. 일반 네트워크 구성

7.5.1. 네트워크 인터페이스 구성 파일 생성

네트워크 스크립트에 어떤 인터페이스가 쓰일지는 일반적으로 /etc/sysconfig/에 있는 파일에 따라 달라진다. 이 디렉토리에는 ifconfig.xyz같은, 구성할 각 인터페이스에 대한 파일이 포함되어야 한다. 여기서 “xyz”는 네트워크 카드를 칭한다. 보통 인터페이스 이름(가령 eth0)이 적합하다. 이 파일 안에는 IP 주소, 서브넷 마스크 등과 같은 해당 인터페이스의 속성이 있다. 파일 이름의 확장자 앞부분은 ifconfig가 되어야 한다.

참고

앞 절의 절차를 따르지 않았다면 Udev는 네트워크 카드 인터페이스 이름을 시스템 물리적 특성에 따라 enp2s1과 같은 식으로 할당할 것이다. 인터페이스 이름을 모르겠다면 시스템을 부팅한 후 언제든지 ip link나 ls /sys/class/net를 실행하면 된다.

다음 명령은 정적 IP 주소를 가진 eth0 장치의 샘플 파일을 생성한다:

```
cd /etc/sysconfig/
cat > ifconfig.eth0 << "EOF"
ONBOOT=yes
IFACE=eth0
SERVICE=ipv4-static
IP=192.168.1.2
GATEWAY=192.168.1.1
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

기울임꼴로 표시된 값은 적절한 설정과 맞도록 모든 파일에서 수정해야 한다.

ONBOOT 변수가 “yes”로 설정되면 System V 네트워크 스크립트는 네트워크 인터페이스 카드(NIC)를 시스템 부팅 중에 불러올 것이다. “yes” 이외의 항목으로 설정되면 NIC는 네트워크 스크립트에서 무시되고 자동으로 잡히지 않는다. 인터페이스는 ifup과 ifdown 명령으로 수동으로 시작하거나 중지할 수 있다.

IFACE 변수는 인터페이스 이름(예: eth0)을 정의한다. 이는 모든 네트워크 장치 구성 파일에 필요하다. 파일명의 확장자는 반드시 이 값과 일치해야 한다.

SERVICE 변수는 IP 주소를 얻기 위해 사용되는 방법을 정의한다. LFS-Bootscipts 패키지의 IP 할당 방식은 모듈형이며, /lib/services/ 디렉토리에 추가 파일을 만들면 다른 IP 할당 방법을 사용할 수 있다. 이는 BLFS 책에서 다루는 DHCP(Dynamic Host Configuration Protocol)에 일반적으로 사용된다.

GATEWAY 변수는 기본 게이트웨이 IP 주소가 존재한다면 그것을 포함해야 한다. 존재하지 않는다면 이 변수 전체를 주석 처리하라.

PREFIX 변수는 서브넷에서 사용되는 비트 수를 사용한다. IP 주소의 각 옥텟은 8비트이다. 서브넷의 넷마스크가 255.255.255.0이라면, 네트워크 번호를 지정하는 데에 처음 3개의 옥텟(24비트)을 사용한다. 넷마스크가 255.255.255.240이라면, 처음 28비트를 사용할 것이다. 보통 24비트보다 긴 접두어(prefix)는 DSL과 케이블 기반 인터넷 서비스 공급자(ISP)가 사용한다. 이 예(접두어=24)에서 넷마스크는 255.255.255.0이다. 특정 서브넷에 따라 PREFIX 변수를 조정하라. 생략할 경우 접두사는 기본적으로 24로 지정된다.

자세한 정보는 ifup man 페이지를 참고하라.

7.5.2. /etc/resolv.conf 파일 생성

인터넷 도메인 이름을 IP 주소로, 또는 그 반대로 변환하려면 시스템에서 DNS(Domain Name Service) 이름을 확인할 방법이 필요하다. ISP나 네트워크 관리자가 사용할 수 있는 DNS 서버의 IP 주소를 /etc/resolv.conf에 두는 것이 가장 좋다. 다음을 실행하여 파일을 작성하라:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf

domain <도메인 이름>
nameserver <주 네임서버의 IP 주소>
nameserver <보조 네임서버의 IP 주소>

# End /etc/resolv.conf
EOF
```

domain 문은 생략하거나 search 문으로 바꿀 수 있다. 자세한 내용은 resolv.conf의 man 페이지를 참고하라.

<네임서버의 IP 주소>를 설정에 가장 적합한 DNS의 IP 주소로 대체하라. 입력 항목은 보통 두 개 이상일 것이다(예비 용도를 위한 보조 서버 필요). DNS 서버가 단 하나만 필요하거나 원한다면 파일에서 두 번째 nameserver 행을 제거하라. 로컬 네트워크에 속한 라우터의 IP 주소를 사용할 수도 있다.

참고

구글 공용 IPv4 DNS 주소는 8.8.8.8과 8.8.4.4이다.

7.5.3. 시스템 호스트 이름 설정

부팅 중에 /etc/hostname 파일이 시스템의 호스트 이름을 설정하는 데 사용된다.

/etc/hostname 파일을 생성하고 다음을 실행하여 호스트 이름을 입력하라:

```
echo "<lfs>" > /etc/hostname
```

<lfs>를 컴퓨터에 부여된 이름으로 바꿔야 한다. 여기에 FQDN(Fully Qualified Domain Name)을 입력하지 않도록 하라. 그것은 /etc/hosts 파일에 저장한다.

7.5.4. /etc/hosts 파일 사용자 정의

/etc/hosts 파일에서 사용할 IP 주소, FQDN 및 가능한 별칭을 결정하라. 문법은 다음과 같다:

```
IP_address myhost.example.org aliases
```

컴퓨터가 인터넷에 공개되어있지 않는 한(즉, 등록된 도메인과 할당된 IP 주소의 유효한 블록이 있지 않는 한—대부분의 사용자는 해당되지 않음), IP 주소가 사설 네트워크 IP 주소 범위에 있도록 하라. 유효한 범위는 다음과 같다:

```
사설 네트워크 주소 범위      일반 접두어
10.0.0.1 - 10.255.255.254          8
172.x.0.1 - 172.x.255.254         16
192.168.y.1 - 192.168.y.254        24
```

x는 16-31 사이의 임의의 숫자이며, y는 0-255 사이의 임의의 숫자이다.

유효한 사설 IP 주소는 192.168.1.1을 쓸 수 있다. 이 IP에 유효한 FQDN은 lfs.example.org이 될 것이다.

네트워크 카드를 사용하지 않더라도 유효한 FQDN이 필요하다. 이는 특정 프로그램이 올바르게 작동하기 위함이다.

다음을 실행하여 /etc/hosts 파일을 생성하라:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts

127.0.0.1 localhost
127.0.1.1 <FQDN> <HOSTNAME>
<192.168.1.1> <FQDN> <HOSTNAME> [alias1] [alias2 ...]
::1       localhost ip6-localhost ip6-loopback
ff02::1   ip6-allnodes
ff02::2   ip6-allrouters

# End /etc/hosts
EOF
```

<192.168.1.1>, <FQDN>, <HOSTNAME> 값은 특정 용도나 요구 사항(네트워크/시스템 관리자가 IP 주소를 할당하고 컴퓨터가 기존 네트워크에 연결될 경우)에 맞게 변경해야 한다. 선택 사항인 별칭 이름은 생략할 수 있다.

7.6. System V 부트스크립트 사용 및 설정

7.6.1. System V 부트스크립트의 작동 방식

리눅스는 run-levels라는 개념을 기반으로 하는 SysVinit이라는 특수 부팅 기능을 사용한다. 이는 시스템마다 상당히 다를 수 있기 때문에, 특정 리눅스 배포판에서 잘 작동했다고 해서 LFS에서도 똑같이 작동하리라 보장할 수 없다. LFS는 나름대로의 방법이 있지만, 보편적으로 받아들여지는 기준을 존중한다.

SysVinit(이제부터는 “init”으로 지칭한다)는 run-levels 체계를 사용하여 동작한다. run-levels는 0부터 6까지 7개이고(실제로는 더 많지만, 특별한 경우를 위한 것으로 일반적으로는 잘 사용되지 않는다. 자세한 내용은 init(8)를 참고하라), 각 레벨은 컴퓨터가 시작될 때 수행해야 하는 작업에 해당된다. 기본 run-level은 3이다. 다음은 구현된 다양한 run-level에 대한 설명이다:

```
0: 컴퓨터를 종료함
1: 싱글 유저 모드
2: 네트워크 미지원 멀티 유저 모드
3: 네트워크 지원 멀티 유저 모드
4: 사용자 정의를 위해 보류되어 있음, 사용자 정의되지 않았다면 3과 동일하게 작동
5: 4와 동일, 보통 GUI 로그인에 사용됨(X의 xdm이나 KDE의 kdm)
6: 컴퓨터를 재시작함
```

7.6.2. Sysvinit 설정

커널 초기화 중에, 실행되는 첫 번째 프로그램은 명령줄에 따로 지정되지 않는 한 기본적으로 init이다. 이 프로그램은 초기화 파일인 /etc/inittab를 읽어들인다. 다음을 실행하여 이 파일을 생성하라:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc S

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# End /etc/inittab
EOF
```

이 초기화 파일에 대한 설명은 inittab의 man 페이지에 있다. LFS의 경우 실행 명령은 rc이다. 위의 초기화 파일은 /etc/rc.d/rcS.d 디렉토리에서 S로 시작하는 모든 스크립트와, /etc/rc.d/rc?.d 디렉토리(물음표는 initdefault 값으로 대체)에서 S로 시작하는 모든 스크립트를 실행하도록 rc에 지시한다.

편의상 rc 스크립트는 /lib/lsb/init-functions의 함수 라이브러리를 읽어 들인다. 이 라이브러리는 선택적 구성 파일인 /etc/sysconfig/rc.site도 읽어 들인다. 아래에서 설명하는 어떤 시스템 구성 파일 매개 변수든 이 파일에 배치해서 이 파일 하나에 모든 시스템 매개 변수를 통합할 수 있다.

디버깅 상의 편의를 위해, functions 스크립트는 모든 출력을 /run/var/bootlog에 기록한다. /run 디렉토리는 tmpfs이므로 이 파일은 시스템 종료나 재부팅 시에는 보존되지 않지만, 부팅 절차가 끝날 때 영구적인 /var/log/boot.log 파일에 내용이 첨부된다.

7.6.2.1. Run Levels 변경

run-levels 변경은 init <runlevel>을 통해 이루어지며, 여기서 <runlevel>은 대상 run-level이다. 예를 들어, 시스템을 재부팅하려면 사용자는 reboot 명령의 별칭인 init 6 명령을 사용할 수 있다. 마찬가지로, init 0은 halt 명령의 별칭이다.

/etc/rc.d 아래에는 rcsysinit.d와 rc?.d(여기서 ? 는 run-level의 번호)같은 디렉토리 등의 여러 디렉토리가 있으며, 모두 다수의 심볼릭 링크를 포함하고 있다. 어떤 것은 K로 시작하고, 다른 일부는 S로 시작하며, 그들 모두는 첫 글자에 이어 두 개의 숫자를 갖고 있다. K는 서비스를 중지(종료)하고 S는 서비스를 시작하는 것을 의미한다. 숫자는 00에서 99까지 스크립트가 실행되는 순서를 결정한다—숫자가 낮을수록 더 빨리 실행된다. init이 다른 run-level로 전환되면, 선택한 runlevel에 따라 적절한 서비스가 시작되거나 중지된다.

실제 스크립트들은 /etc/rc.d/init.d에 있다. 실제로 동작하는 것은 그것들이고 심볼릭 링크들은 모두 그것들을 가리킨다. K 링크와 S 링크는 /etc/rc.d/init.d의 동일한 스크립트를 가리킨다. 이는 start, stop, restart, reload, 및 status와 같은 다른 매개 변수를 사용하여 스크립트를 호출할 수 있기 때문이다. K 링크가 호출되면 적절한 스크립트가 stop 인수와 함께 실행된다. S 링크가 호출되면 start 인수와 함께 적절한 스크립트가 실행된다.

이 설명에는 한 가지 예외가 있다. rc0.d와 rc6.d 디렉토리의 S로 시작하는 링크들은 어떤 것도 시작시키지 않는다. 그것들은 무언가를 중단시키기 위한 매개 변수 stop과 함께 호출된다. 그 이유는 사용자가 시스템을 재부팅하거나 종료할 때는 아무것도 시작할 필요가 없기 때문이다. 시스템은 오로지 중단되기만 하면 된다.

다음은 스크립트가 인수에 따라 수행하는 작업에 대한 설명이다:

start
: 서비스가 시작된다.

stop
: 서비스가 중지된다.

restart
: 서비스가 중지되었다가 다시 시작된다.

reload
: 서비스 구성이 업데이트된다. 이는 서비스의 구성 파일이 수정된 후 서비스를 재시작할 필요가 없을 때 사용된다.

status
: 서비스가 실행 중인지, PID가 몇인지 출력한다.

부팅 프로세스가 작동하는 방식을 자유롭게 수정하라(어떻게 하든 간에 결국 당신만의 LFS 시스템이다). 여기서 주어진 파일들은 이를 수행하는 방법의 예시이다.

7.6.3. Udev 부트스크립트

/etc/rc.d/init.d/udev initscript는 udevd를 시작하고, 커널에 의해 이미 생성된 "coldplug" 장치를 작동시키며 모든 규칙이 처리 완료될 때까지 기다린다. 또 스크립트는 /sbin/hotplug 의 기본값으로부터 uevent 핸들러를 해제(unset)한다. 이는 커널이 더 이상 외부 바이너리를 호출할 필요가 없기 때문이다. 대신 udevd가 커널에서 발생하는 uevent를 netlink 소켓에서 수신한다.

/etc/rc.d/init.d/udev_retry initscript는 mountfs 스크립트가 실행되고도 마운트되지 않은 파일 시스템이 필요한 규칙이 있는 하위 시스템에 대한 이벤트를 다시 실행(re-triggering)하는 작업을 한다(특히 /usr와 /var가 이런 문제를 일으킬 수 있다). 이 스크립트는 mountfs 스크립트 이후에 실행되므로, 그러한 규칙들은 (재실행된다면) 두 번째 시도부터는 성공적으로 작동할 것이다. 이는 /etc/sysconfig/udev_retry 파일로부터 구성된다; 이 파일에서 주석 이외의 모든 단어는 재시도 시 실행할 하위 시스템 이름으로 간주된다. 장치의 하위 시스템을 찾으려면 udevadm info --attribute-walk <device>를 사용하라. 여기서 <device>는 /dev/sr0 또는 /sys/class/rtc와 같이 /dev나 /sys의 절대 경로이다.

커널 모듈 로드와 udev에 대한 정보는 7.3.2.3절. “모듈 로딩”를 참고하라.

7.6.4. 시스템 시계 설정

setclock 스크립트는 BIOS 또는 CMOS(Complementary Metal Oxide Semiconductor) 시계라고도 하는 하드웨어 시계로부터 시간을 읽어 들인다. 하드웨어 시계가 UTC로 설정된 경우, 이 스크립트는 /etc/localtime 파일(사용자가 속해 있는 시간대를 hwclock 프로그램에 알려주는 파일)을 사용하여 하드웨어 시계의 시간을 로컬 시간으로 변환한다. 하드웨어 클럭이 UTC로 설정되어 있는지 여부는 달리 확인할 방법이 없으므로 수동으로 설정해야 한다.

setclock은 커널이 부팅 시 하드웨어 기능을 감지할 때 udev를 통해 실행된다. stop 매개 변수와 함께 수동으로 실행하여 시스템 시간을 CMOS 시계에 저장할 수도 있다.

하드웨어 시계가 UTC로 설정되어 있는지의 여부를 기억할 수 없다면 hwclock --localtime --show 명령을 실행하여 확인하라. 이는 하드웨어 시계에 따라 현재 시간을 표시한다. 만약 이 시간이 당신의 다른 시계와 일치하면, 하드웨어 시계는 현지 시간으로 설정된 것이다. hwclock의 출력이 현지 시간이 아니라면 UTC 시간으로 설정됐을 가능성이 있다. hwclock에서 표시된 시간에 적절한 시간대를 더하거나 빼서 확인해보라. 예를 들어 현재 GMT -0700 이라고도 하는 MST 표준 시간대에 있다면, 현지 시간에 7시간을 더하라.

하드웨어 시계가 UTC 시간으로 설정되지 않았다면 아래의 UTC 변수의 값을 0(zero)으로 변경하라.

다음을 실행하여 새 /etc/sysconfig/clock 파일을 생성하라:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock

UTC=1

# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=

# End /etc/sysconfig/clock
EOF
```

LFS에서 시간을 다루는 방식을 설명하는 좋은 힌트들을 http://www.linuxfromscratch.org/hints/downloads/files/time.txt에서 확인할 수 있다. 시간대, UTC, TZ 환경 변수와 같은 주제들을 설명한다.

참고

CLOCKPARAMS와 UTC 매개 변수는 /etc/sysconfig/rc.site 파일에서 대신 설정할 수 있다.

7.6.5. 리눅스 콘솔 설정

이 절에서는 키보드 맵, 콘솔 글꼴 및 콘솔 커널 로그 레벨을 설정하는, console 부트스크립트를 설정하는 방법을 설명한다. ASCII 문자가 아닌 문자(예: 저작권 기호, 영국 파운드 기호 및 유로 기호)를 사용하지 않으며 키보드가 미국식 키보드라면 이 절의 많은 부분을 건너뛸 수 있다. 설정 파일이 없으면,(또는 rc.site의 같은 설정), console 부트스크립트는 아무 작업도 수행하지 않는다.

console 스크립트는 /etc/sysconfig/console 파일로부터 설정 정보를 읽어들인다. 사용할 키맵과 화면 글꼴을 결정하라. 다양한 언어별 HOWTO도 도움이 될 수 있다(http://www.tldp.org/HOWTO/HOWTO-INDEX/other-lang.html를 참고하라). 결정하기 힘들다면 /usr/share/keymaps와 /usr/share/consolefonts 디렉토리에서 유효한 키맵과 화면 글꼴을 확인하라. loadkeys(1) 및 setfont(8) 매뉴얼 페이지를 읽고 이러한 프로그램에 대한 올바른 인수를 결정하라.

/etc/sysconfig/console 파일은 다음과 같은 형식의 행을 포함한다: VARIABLE="value". 다음과 같은 변수들이 인식된다:

LOGLEVEL
: 이 변수는 dmesg에서 설정한 대로 콘솔로 전달되는 커널 메시지의 로그 수준을 지정한다. 유효한 수준은 "1"(메시지 없음)에서 "8"까지이다. 기본값은 "7"이다.

KEYMAP
: 이 변수는 대표적으로 로드할 키맵의 이름(예: “it”)과 같은, loadkeys 프로그램에 대한 인수를 지정한다. 이 변수를 설정하지 않으면 부트스크립트가 loadkeys 프로그램을 실행하지 않고 기본 커널 키맵이 사용된다. 키맵 중 몇 가지는 동일한 이름의 여러 버전이 있다(qwerty/와 qwertz/에는 cz와 그 변형들, olpc/와 qwerty/에는 es, fgGIod/와 qwerty/에는 trf). 이 경우 적절한 키맵이 로드되도록 상위 디렉토리(예: qwerty/es)도 지정해야 한다.

KEYMAP_CORRECTIONS
: 이 (드물게 쓰이는) 변수는 loadkeys 프로그램을 재호출하는 데에 쓰이는 인수를 지정한다. 이는 기본 키맵이 완전히 만족스럽지 않아서 약간 조정하고자 할 때 유용하다. 예를 들어 유로 기호를 기본적으로 포함하지 않는 키맵에 유로 기호를 포함시키려면 이 변수를 “euro2”로 설정하라.

FONT
: 이 변수는 setfont 프로그램에 대한 인수를 지정한다. 이는 일반적으로 글꼴 이름과 “-m”, 그리고 로드할 응용 문자맵의 이름으로 구성된다. 가령 “lat1-16” 글꼴을 “8859-1” 응용 문자맵(미국에서 적합하다)과 함께 로드하려면, 이 변수를 “lat1-16 -m 8859-1”로 설정하라. UTF-8 모드에서 커널은 키맵에 구성된 8비트 키코드를 UTF-8로 전환하기 위해 응용 문자맵을 사용한다. 따라서 "-m" 매개 변수의 인수는 키맵에 구성된 키코드의 인코딩으로 설정되어야 한다.

UNICODE
: 콘솔을 UTF-8 모드로 전환하려면 이 변수를 “1”, “yes” 또는 “true”로 설정하라. 이는 UTF-8 기반 로케일에서 사용 가능하며 UTF-8 기반 로케일이 아닐 경우엔 문제를 일으킨다.

LEGACY_CHARSET
: 많은 키보드 레이아웃은 Kbd 패키지에 기본 유니코드 키맵이 없다. console 부트스크립트는 이 변수가 사용 가능한 비 UTF-8 키맵의 인코딩으로 설정되면 사용 가능한 키맵을 UTF-8로 즉시 변환한다.

몇 가지 예시:

- 유니코드가 아닌 설정에서는 일반적으로 KEYMAP과 FONT 변수만 있으면 된다. 가령 폴란드어 설정은 다음과 같이 할 수 있다:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="pl2"
FONT="lat2a-16 -m 8859-2"

# End /etc/sysconfig/console
EOF
```

- 위에서 언급한 바와 같이, 가끔은 기본 키맵을 약간 조정해야할 때가 있다. 다음 예시는 독일어 키맵에 유로 기호를 추가한다:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
FONT="lat0-16 -m 8859-15"
UNICODE="1"

# End /etc/sysconfig/console
EOF
```

- 다음은 기본 UTF-8 키맵이 존재하는, 유니코드가 사용 가능한 불가리아어 예시이다:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="LatArCyrHeb-16"

# End /etc/sysconfig/console
EOF
```

- 이전 예시에서 512-glyph LatArCyrHeb-16 글꼴을 사용했기 때문에, 프레임 버퍼를 사용하지 않는 한 리눅스 콘솔에서 밝은 색상은 더 이상 사용할 수 없다. 프레임버퍼 없이 밝은 색을 원하며 자신의 언어에 속하지 않는 문자를 사용하지 않아도 된다면, 아래와 같이 언어별 256-glyph 글꼴을 사용하는 것도 가능하다:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="bg_bds-utf8"
FONT="cyr-sun16"

# End /etc/sysconfig/console
EOF
```

- 다음 예제는 ISO-8859-15에서 UTF-8로의 키맵 자동 변환과 유니코드 모드에서 데드키를 활성화 하는 법을 보여준다:

```
cat > /etc/sysconfig/console << "EOF"
# Begin /etc/sysconfig/console

UNICODE="1"
KEYMAP="de-latin1"
KEYMAP_CORRECTIONS="euro2"
LEGACY_CHARSET="iso-8859-15"
FONT="LatArCyrHeb-16 -m 8859-15"

# End /etc/sysconfig/console
EOF
```

- 일부 키맵들은 데드키(즉, 스스로 문자를 생성하지 않고 그 뒤에 입력된 문자에 액센트를 붙이는 키)가 있거나 구성 규칙을 가진다(예: 기본 키맵에서 “Æ를 입력하기 위해 Ctrl+. A E를 누름”). 리눅스-5.5.3은 함께 합성할 원본 문자가 멀티바이트가 아닐 때만 키맵의 데드 키와 구성 규칭을 올바르게 해석한다. 이 결함은 유럽 언어들의 키맵에는 영향을 미치지 않는데, 왜냐하면 그 악센트들은 악센트가 없는 ASCII 문자에 추가되거나 두 개의 ASCII 문자가 함께 결합되기 때문이다. 그러나 UTF-8 모드, 예를 들어 그리스어에서는, 문자 “alpha”에 악센트를 붙여야 하는 문제가 있다. 해결책은 UTF-8의 사용을 피하거나, 입력 처리에서 이러한 제한이 없는 X 윈도 시스템을 설치하는 것이다.
- 한국어, 일본어, 중국어를 비롯한 일부 언어들은 리눅스 콘솔이 해당 언어를 표시하도록 구성할 수 없다. 이러한 언어가 필요한 사용자는 X 윈도 시스템, 필요한 문자 범위를 포함하는 글꼴과 적절한 입력기(예를 들어 SCIM, 다양한 언어를 지원)를 설치해야 한다.

참고

/etc/sysconfig/console 파일은 리눅스 텍스트 콘솔 언어만을 제어한다. X 윈도 시스템에서의 적절한 키보드 레이아웃이나 터미널 글꼴 설정, 또는 ssh 세션이나 시리얼 콘솔과는 아무 관련이 없다. 이런 상황에서는 위에서 마지막으로 언급한 두 제한 사항이 해당되지 않는다.

7.6.6. 부팅 시 파일 생성

가끔은 부팅 시에 파일을 생성하고 싶을 수도 있다. 예를 들어 /tmp/.ICE-unix 디렉토리가 필요할 수 있다. 이 작업은 /etc/sysconfig/createfiles 구성 스크립트에 항목을 생성하여 수행할 수 있다. 이 파일의 작성 양식은 기본 구성 파일의 주석에 적혀있다.

7.6.7. sysklogd 스크립트 구성

sysklogd 스크립트는 System V 초기화 중 syslogd 프로그램을 호출한다. -m 0 옵션은 syslogd가 로그 파일에 기본적으로 20분마다 주기적으로 기록하는 타임스탬프 표시를 비활성화한다. 이 주기적 타임스탬프 표시를 활성화하려면, /etc/sysconfig/rc.site의 SYSKLOGD_PARMS 변수를 원하는 값으로 정의하라. 예를 들어, 모든 매개 변수를 제거하려면 변수를 빈 값으로 설정하라:

```
SYSKLOGD_PARMS=
```

더 많은 옵션은 man syslogd를 참고하라.

7.6.8. rc.site 파일

선택 사항인 /etc/sysconfig/rc.site 파일에는 각 SystemV 부트 스크립트에 대해 자동으로 설정된 설정이 포함되어 있다. 이 파일은 /etc/sysconfig/ 디렉토리의 hostname, console, 그리고 clock 파일에 지정된 값을 대체할 수 있다. 관련 변수들이 이 별도의 파일들과 rc.site에 모두 존재할 경우, 스크립트별 파일의 값들은 덮여 쓰인다.

rc.site에는 부팅 절차의 다른 부분을 사용자 정의할 수 있는 매개 변수도 포함되어 있다. IPROMPT 변수를 설정하면 부트스크립트를 선택적으로 실행할 수 있다. 다른 옵션은 파일 주석에 설명되어 있다. 파일의 기본 버전은 다음과 같다:

```
# rc.site
# Optional parameters for boot scripts.

# Distro Information
# These values, if specified here, override the defaults
#DISTRO="Linux From Scratch" # The distro name
#DISTRO_CONTACT="lfs-dev@linuxfromscratch.org" # Bug report address
#DISTRO_MINI="LFS" # Short name used in filenames for distro config

# Define custom colors used in messages printed to the screen

# Please consult `man console_codes` for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles

# These values, if specified here, override the defaults
#BRACKET="₩₩033[1;34m" # Blue
#FAILURE="₩₩033[1;31m" # Red
#INFO="₩₩033[1;36m"    # Cyan
#NORMAL="₩₩033[0;39m"  # Grey
#SUCCESS="₩₩033[1;32m" # Green
#WARNING="₩₩033[1;33m" # Yellow

# Use a colored prefix
# These values, if specified here, override the defaults
#BMPREFIX="     "
#SUCCESS_PREFIX="${SUCCESS}  *  ${NORMAL} "
#FAILURE_PREFIX="${FAILURE}*****${NORMAL} "
#WARNING_PREFIX="${WARNING} *** ${NORMAL} "

# Manually seet the right edge of message output (characters)
# Useful when resetting console font during boot to override
# automatic screen width detection
#COLUMNS=120
```

```
# Interactive startup
#IPROMPT="yes" # Whether to display the interactive boot prompt
#itime="3"    # The amount of time (in seconds) to display the prompt

# The total length of the distro welcome string, without escape codes
#wlen=$(echo "Welcome to ${DISTRO}" | wc -c )
#welcome_message="Welcome to ${INFO}${DISTRO}${NORMAL}"

# The total length of the interactive string, without escape codes
#ilen=$(echo "Press 'I' to enter interactive startup" | wc -c )
#i_message="Press '${FAILURE}I${NORMAL}' to enter interactive startup"

# Set scripts to skip the file system check on reboot
#FASTBOOT=yes

# Skip reading from the console
#HEADLESS=yes

# Write out fsck progress if yes
#VERBOSE_FSCK=no

# Speed up boot without waiting for settle in udev
#OMIT_UDEV_SETTLE=y

# Speed up boot without waiting for settle in udev_retry
#OMIT_UDEV_RETRY_SETTLE=yes

# Skip cleaning /tmp if yes
#SKIPTMPCLEAN=no

# For setclock
#UTC=1
#CLOCKPARAMS=

# For consolelog (Note that the default, 7=debug, is noisy)
#LOGLEVEL=7

# For network
#HOSTNAME=mylfs

# Delay between TERM and KILL signals at shutdown
#KILLDELAY=3

# Optional sysklogd parameters
#SYSKLOGD_PARMS="-m 0"

# Console parameters
```

```
#UNICODE=1
#KEYMAP="de-latin1"
#KEYMAP_CORRECTIONS="euro2"
#FONT="lat0-16 -m 8859-15"
#LEGACY_CHARSET=
```

7.6.8.1. 부팅 및 시스템 종료 스크립트 사용자 정의

LFS 부트 스크립트는 상당히 효율적인 방식으로 시스템을 부팅하고 종료하지만, rc.site 파일을 약간 수정해서 속도를 더욱 향상시키고 사용자의 취향에 따라 메시지를 조정할 수 있는 방법이 몇 가지 있다. 이렇게 하려면 위의 /etc/sysconfig/rc.site 파일에서 설정을 조정하라.

- 부트 스크립트 udev에는 완료하는 데 약간의 시간이 걸리는 udev settle 호출이 있다. 시스템에 있는 장치에 따라 시간이 걸리거나 걸리지 않을 수 있다. 단순 파티션과 단일 이더넷 카드만 있는 경우 부팅 프로세스는 이 명령을 기다릴 필요가 없다. 이를 건너 뛰려면 변수를 다음과 같이 설정하라: OMIT_UDEV_SETTLE=y
- 부트 스크립트 udev_retry도 마찬가지로 udev settle을 기본적으로 실행한다. 이 명령은 /var 디렉토리가 별도로 마운트된 경우에만 필요하다. 시계에 /var/lib/hwclock/adjtime 파일이 필요하기 때문이다. 다른 사용자 정의도 udev가 완료되기를 기다려야 하지만, 대부분은 불필요하다. 다음과 같이 변수를 설정하여 이 명령을 건너뛴다: OMIT_UDEV_RETRY_SETTLE=y.
- 기본적으로 파일 시스템 검사는 암묵적으로 진행되기 때문에 부팅 프로세스가 지연되는 것처럼 보일 수 있다. fsck 출력을 활성화하려면 다음과 같이 변수를 설정하라: VERBOSE_FSCK=y.
- 재부팅할 때 파일 시스템 검사(fsck)를 건너뛰려면 /fastboot 파일을 만들거나, /sbin/shutdown -f -r now 명령으로 시스템을 재부팅하라. 반면에 /forcefsck를 만들거나 -f 매개 변수 대신 -F와 함께 shutdown 명령을 실행하면 모든 파일 시스템을 강제로 검사할 수도 있다.

  FASTBOOT=y로 설정하면 이 옵션을 지우지 않는 한 부팅 과정에서 fsck를 비활성화한다. 이는 영구적인 설정으로는 권장하지 않는다.
- 일반적으로 /tmp 디렉토리의 모든 파일은 부팅 시 삭제된다. 이는 존재하는 파일이나 디렉토리의 수에 따라 부팅 과정이 현저하게 지연될 수 있다. 이러한 파일 제거를 건너뛰려면 다음과 같이 설정하라: SKIPTMPCLEAN=y.
- 시스템 종료 중 init 프로그램은 시작된 각 프로그램(예: agetty)에 TERM 신호를 보내고 설정된 시간(기본값 3초)을 기다린 뒤, 각 프로세스에 KILL 신호를 보낸 후 다시 기다린다. 이 과정은 각자 고유한 스크립트에 의해 종료되지 않은 모든 프로세스들을 대상으로 sendsignals 스크립트에 의해 반복된다. init의 대기 시간은 매개 변수로 조절할 수 있다. 예를 들어 init의 대기 시간을 없애려면, 시스템 종료나 재부팅할 때 -t0 매개 변수를 사용하라(예를 들면 /sbin/shutdown -t0 -r now). sendsignals 스크립트의 대기 시간은 다음과 같이 매개 변수를 설정하여 건너뛸 수 있다: KILLDELAY=0.

7.7. Bash 셸 시작 파일

셸 프로그램(이하 “셸”로 칭함)인 /bin/bash는 실행할 환경을 만들기 위해 시작 파일 모음을 사용한다. 각 파일들은 특정한 용도를 가지며 로그인과 대화형 환경에 따라 다른 영향을 미친다. /etc 디렉토리의 파일은 전역 설정을 제공한다. 홈 디렉토리에 동일한 파일이 있다면, 전역 설정보다 우선시된다.

/bin/login을 통해 /etc/passwd 파일을 읽어 로그인에 성공하면 대화형 로그인 셸이 시작된다. 대화형 비로그인 셸은 명령줄에서 시작한다(예: [prompt]$/bin/bash). 비대화형 셸은 일반적으로 셸 스크립트가 실행 중일 때 나타난다. 스크립트를 처리하느라 명령들 간에 사용자의 입력을 기다리지 않기 때문에 비대화형이라 한다.

더 많은 정보는 info bash의 Bash Startup Files and Interactive Shells절을 참고하라.

/etc/profile과 ~/.bash_profile 파일들은 셸이 대화형 로그인 셸로 호출될 때 읽힌다.

아래에서 설명할 기본 /etc/profile은 자국어 지원에 필요한 몇 가지 환경 변수를 설정한다. 이들을 적절히 설정하면 다음과 같은 결과들을 얻을 수 있다:

- 프로그램들의 출력을 자국어로 출력
- 문자를 문자, 숫자 및 기타 기호로 올바르게 분류한다. 이는 bash가 영미권 이외의 비 ASCII문자를 명령줄에서 올바르게 다루기 위해 필요하다.
- 국가에 대한 올바른 알파벳 정렬 순서
- 적절한 기본 용지 크기
- 통화, 시간 및 날짜 값의 올바른 형식 지정

아래 <ll>을 원하는 언어에 대한 2글자 코드로(예: “en”), <CC>를 해당 국가(예: “GB”)로 대체하라. <charmap>은 선택한 지역에 대한 표준 문자표로 대체해야 한다. “@euro”와 같은 선택적 옵션도 사용할 수 있다.

Glibc에서 지원하는 모든 국가/지역(locale)의 목록은 다음 명령을 실행하여 얻을 수 있다:

```
locale -a
```

문자표는 많은 별칭을 가질 수 있다. 예를 들어 “ISO-8859-1”은 “iso8859-1” 또는 “iso88591”이라고도 한다. 일부 응용 프로그램들은 이 다양한 동의어들을 올바르게 처리하지 못하므로(가령 “UTF-8”은 “utf8”이 아닌 “UTF-8”로 표기되어야 한다), 대부분의 경우 특정 국가/지역의 정식 명칭을 선택하는 것이 가장 안전하다. 정식 명칭을 알아내려면 다음 명령을 실행하라. 여기서 <locale name>은 선호하는 국가/지역에 대한 locale -a의 출력 결과이다 (이 예시에서는 “en_GB.iso88591”).

```
LC_ALL=<locale name> locale charmap
```

“en_GB.iso88591” 로케일에 대해서, 위 명령은 다음과 같이 출력한다:

```
ISO-8859-1
```

이렇게 하면 최종 국가/지역 설정은 “en_GB.ISO-8859-1”이 된다. 위의 방법을 사용하여 찾은 로케일을 Bash 시작 파일에 추가하기 전에 먼저 테스트하는 것이 중요하다:

```
LC_ALL=<locale name> locale language
LC_ALL=<locale name> locale charmap
LC_ALL=<locale name> locale int_curr_symbol
LC_ALL=<locale name> locale int_prefix
```

위의 명령은 언어 이름, 해당 국가에서 사용하는 문자 인코딩, 현지 통화 및 국제 전화 지역 번호 등을 출력한다. 위의 명령 중 하나가 아래 표시된 것과 유사한 메시지와 함께 실패한다면, 해당 로케일이 6장에서 설치되지 않았거나 Glibc의 기본 설치로부터 지원되지 않음을 의미한다.

```
locale: Cannot set LC_* to default locale: No such file or directory
```

이 경우 localedef 명령을 사용하여 원하는 로케일을 설치하거나, 다른 로케일을 선택해야 한다. 아래 지침은 Glibc로부터 이러한 오류 메시지가 뜨지 않는다고 가정한다.

LFS 이외의 일부 패키지들은 선택한 로케일을 지원하지 않을 수도 있다. X 라이브러리(X 윈도 시스템의 일부)가 한 가지 예로, 로케일이 내부 파일의 문자표 이름 중 하나와 정확히 일치하지 않으면 다음 오류 메시지를 출력한다:

```
Warning: locale not supported by Xlib, locale set to C
```

간혹 Xlib는 "iso88591" 대신 "ISO-8859-1"와 같이, 표준 대시와 함께 대문자로 표기된 문자표를 필요로 한다. 로케일 사양의 문자표 부분을 제거하여 적절한 사양을 찾을 수도 있다. 이는 두 로케일에서 locale charmap 명령을 실행하여 확인할 수 있다. 예를 들어, Xlib에서 이 로케일을 인식하도록 하려면 "de_DE.ISO-8859-15@euro"를 "de_DE@euro"로 변경해야 한다.

다른 패키지도 로케일 이름이 기대값에 알맞지 않을 경우 잘못 작동할 수 있다(그러나 오류 메시지를 반드시 출력하지는 않을 수도 있다). 이 경우, 다른 리눅스 배포판들에서 로케일을 지원하는 방법을 조사하면 몇 가지 유용한 정보를 얻을 수 있다.

올바르게 로케일을 설정했다면 /etc/profile 파일을 생성하라:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile

export LANG=<ll>_<CC>.<charmap><@modifiers>

# End /etc/profile
EOF
```

“C”(기본값)와 “en_US”(미국 영어 사용자에게 권장되는 설정) 로케일은 서로 다르다. “C”는 US-ASCII 7비트 문자 집합을 사용하며, 높은 비트가 설정된 바이트를 잘못된 문자로 처리한다. 따라서, 예를 들면 ls 명령은 해당 로케일의 물음표로 그러한 문자를 대체한다. 또 Mutt나 Pine에서 이러한 문자로 메일을 보내려고 하면 RFC 비준수 메시지가 전송된다(발송 메일의 문자 집합은 “알 수 없는 8-bit”로 표시됨). 따라서 8비트 문자가 절대 필요하지 않을 경우에만 “C” 로케일을 사용할 수 있다.

UTF-8 기반 로케일은 일부 프로그램에서 잘 지원되지 않는다. 문서화 작업이 진행 중이며, 가능하면 http://www.linuxfromscratch.org/blfs/view/9.1/introduction/locale-issues.html를 참고하여 이러한 문제를 해결하라.

7.8. /etc/inputrc 파일 생성

inputrc 파일은 사용자가 터미널에서 명령을 입력하는 동안 편집 기능을 제공하는 Readline 라이브러리의 설정 파일이다. 키보드 입력을 특정 작업으로 변환하여 작동한다. Readline은 Bash 및 대부분의 다른 셸 및 기타 여러 응용 프로그램에서 사용된다.

대부분의 사용자는 사용자별 기능이 필요하지 않으므로 아래 명령어는 로그인하는 모든 사용자가 사용하는 전역 /etc/inputrc를 생성한다. 추후 사용자별로 기본값을 재정의해야 할 경우는 매핑을 수정해서 사용자의 홈 디렉토리에 .inputrc 파일을 만들 수 있다.

inputrc 파일을 편집하는 방법에 대한 자세한 내용은 Readline Init File의 info bash를 참고하라. info readline 또한 정보를 얻기 좋다.

아래는 다양한 옵션의 기능을 설명하기 위한 주석과 함께 일반적인 전역 inputrc 예시이다. 주석과 명령은 서로 같은 줄에 있을 수 없다는 점에 유의하라. 다음을 실행하여 파일을 생성한다:

```
cat > /etc/inputrc << "EOF"
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>

# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off

# Enable 8bit input
set meta-flag On
set input-meta On

# Turns off 8th bit stripping
set convert-meta Off

# Keep the 8th bit for display
set output-meta On

# none, visible or audible
set bell-style none

# All of the following map the escape sequence of the value
# contained in the 1st argument to the readline specific functions
"₩eOd": backward-word
"₩eOc": forward-word

# for linux console
"₩e[1~": beginning-of-line
"₩e[4~": end-of-line
"₩e[5~": beginning-of-history
"₩e[6~": end-of-history
"₩e[3~": delete-char
"₩e[2~": quoted-insert

# for xterm
"₩eOH": beginning-of-line
"₩eOF": end-of-line

# for Konsole
"₩e[H": beginning-of-line
"₩e[F": end-of-line

# End /etc/inputrc
EOF
```

7.9. /etc/shells 파일 생성

shells 파일에는 시스템의 로그인 셸 목록이 들어 있다. 응용 프로그램은 이 파일을 사용하여 셸이 유효한지 확인한다. 각 셸에 대해 디렉토리 구조의 루트(/)에 상대적인 셸 경로로 구성된 단일 행이 있어야 한다.

예를 들면, 루트가 아닌 일반 사용자가 자신의 계정에 대한 로그인 셸을 변경할 수 있는지 확인할 때는 chsh이 이 파일을 검토한다. 변경하려는 셸이 목록에 있지 않으면, 사용자의 변경 요청이 거부된다.

이는 /etc/shells을 찾지 못하면 인터페이스를 띄우지 않는 GDM이나, 기본적으로 이 파일에 포함되지 않은 셸로 접근하는 사용자를 차단하는 FTP 데몬과 같은 응용 프로그램들에 필요하다.

```
cat > /etc/shells << "EOF"
# Begin /etc/shells

/bin/sh
/bin/bash

# End /etc/shells
EOF
```

8장. LFS 시스템 부팅 가능하게 만들기

8.1. 도입

이제 LFS 시스템을 부팅 가능하게 만들 차례이다. 이 장에서는 fstab 파일을 생성하고, 새 LFS 시스템을 위한 커널을 빌드하며, GRUB 부트 로더를 설치하여 시작 시 부팅할 LFS 시스템을 선택 가능하도록 만드는 법을 다룬다.

8.2. /etc/fstab 파일 생성

/etc/fstab 파일은 일부 프로그램에서 파일 시스템이 기본적으로 마운트될 위치와 순서, 마운트되기 전에 어떤 파일 시스템을 검사할 지(오류 무결성) 결정하는 데 사용된다. 다음과 같은 새 파일 시스템 테이블을 생성하라:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type     options           dump  fsck
#                                                      order

/dev/<xxx>     /          <fff>    defaults          1     1
/dev/<yyy>     swap       swap     pri=1             0     0
proc           /proc      proc     nosuid,noexec,nodev 0     0
sysfs          /sys       sysfs    nosuid,noexec,nodev 0     0
devpts         /dev/pts   devpts   gid=5,mode=620    0     0
tmpfs          /run       tmpfs    defaults          0     0
devtmpfs       /dev       devtmpfs mode=0755,nosuid  0     0

# End /etc/fstab
EOF
```

<xxx>, <yyy>, <fff>를 sda2, sda5, ext4와 같이 해당 시스템에 적합한 값으로 대체하라. 이 파일의 여섯 항목에 대한 자세한 사항은 man 5 fstab를 참고하라.

MS-DOS나 Windows에 쓰였던 파일 시스템들(예: vfat, ntfs, smbfs, cifs, iso9660, udf)은 파일 이름의 비 ASCII 문자가 올바르게 해석되도록 따로 utf8 옵션이 필요하다. 비-UTF-8 로케일의 경우 iocharset의 값이 로케일의 문자 집합과 동일하게 설정되어야 하며, 커널이 이해할 수 있도록 조정되어야 한다. 이는 관련 문자 집합 정의(커널 구성 시 파일 시스템 -> 기본 언어 지원에서 찾을 수 있다)가 커널과 함께 컴파일되거나 모듈로 빌드된 경우에는 문제가 없다. 하지만 로케일의 문자 집합이 UTF-8이라면, 해당 옵션 iocharset=utf8은 파일 시스템의 대/소문자를 구분한다. 이 문제를 해결하려면 UTF-8 로케일에서 iocharset=utf8 대신 특수 옵션 utf8을 사용하라. vfat 및 smbfs 파일 시스템에는 “codepage” 옵션이 필요하다. 해당 국가의 MS-DOS에서 사용되는 코드 페이지 번호로 설정해야 한다. 예를 들어 USB 플래시 드라이브를 마운트한다면, ru_RU.KOI8-R 사용자는 /etc/fstab에 있는 마운트 행에 다음과 같은 옵션이 필요할 것이다:

```
noauto,user,quiet,showexec,codepage=866,iocharset=koi8r
```

ru_RU.UTF-8 사용자에 대한 해당 옵션 부분은 다음과 같다:

```
noauto,user,quiet,showexec,codepage=866,utf8
```

iocharset은 iso8859-1을 위한 기본값이며(파일 시스템이 대/소문자를 구분하지 않게끔 한다), utf8 옵션은 커널에 UTF-8을 사용하여 파일 이름을 변환하도록 지시하여 UTF-8 로케일에서 해석할 수 있도록 한다.

커널 구성 중에 일부 파일 시스템을 위한 기본 코드 페이지 및 iocharset 값을 지정할 수도 있다. 관련 매개 변수의 이름은 “기본 NLS 옵션” (CONFIG_NLS_DEFAULT), “기본 원격 NLS 옵션” (CONFIG_SMB_NLS_DEFAULT), “FAT을 위한 기본 코드페이지” (CONFIG_FAT_DEFAULT_CODEPAGE), 그리고 “FAT을 위한 기본 iocharset” (CONFIG_FAT_DEFAULT_IOCHARSET)이다. 커널 컴파일 타임에는 ntfs 파일 시스템에 대해 이러한 설정을 할 수 있는 방법이 없다.

일부 하드 디스크들은 정전 시에도 ext3 파일 시스템을 안정적으로 유지하도록 만들 수 있다. 이렇게 하려면 /etc/fstab의 적절한 항목에 barrier=1 마운트 옵션을 추가하라. 디스크 드라이브가 이 옵션을 지원하는지 확인하려면 해당 디스크 드라이브에서 hdparm을 실행하라. 예를 들어:

```
hdparm -I /dev/sda | grep NCQ
```

위 명령이 어떤 결과든 출력한다면 옵션이 지원되는 것이다.

참고: 논리적 볼륨 관리 기반 파티션(Logical Volume Management, LVM)은 barrier 옵션을 지원하지 않는다.

8.3. 리눅스-5.5.3

이 리눅스 패키지는 리눅스 커널을 포함하고 있다.

예상 빌드 시간: 4.4 - 66.0 SBU (일반적으로 약 6 SBU)
필요 디스크 공간: 960 - 4250 MB (일반적으로 약 1100 MB)

8.3.1. 커널의 설치

커널 빌드는 설정, 컴파일, 설치 등 몇 단계를 거쳐야 한다. 이 책의 커널을 구성하는 다른 방법은 커널 소스 트리의 README 파일을 참고하라.

다음 명령을 실행하여 컴파일을 준비한다:

```
make mrproper
```

이는 기존 컴파일 설정을 제거하여 커널 트리를 완전히 깨끗하게 한다. 커널 팀은 커널 컴파일 전에 매번 이 명령을 실행할 것을 권장한다. tar 아카이브를 이제 막 해제했다고 해서 깨끗한 상태인 것으로 믿어선 안 된다.

메뉴 기반 인터페이스를 통해 커널을 구성하라. 커널 구성에 대한 일반적인 정보는 http://www.linuxfromscratch.org/hints/downloads/files/kernel-configuration.txt를 참고하라. BLFS는 http://www.linuxfromscratch.org/blfs/view/9.1/longindex.html#kernel-config-index에서 LFS 외부 패키지의 특정 커널 구성 요구 사항에 대한 정보를 제공한다. 커널 구성 및 빌드에 대한 추가 정보는 http://www.kroah.com/lkn/에서 찾을 수 있다.

참고

make defconfig를 실행하여 커널 구성을 시작하는 것이 좋다. 이렇게 하면 기본 구성이 현재 시스템 아키텍쳐를 고려해서 양호한 상태로 설정된다.

다음 기능을 활성화/비활성화/설정하라. 그렇지 않으면 시스템이 제대로 작동하지 않거나 전혀 부팅되지 않을 수 있다:

```
Device Drivers  --->
  Generic Driver Options  --->
   [ ] Support for uevent helper [CONFIG_UEVENT_HELPER]
   [*] Maintain a devtmpfs filesystem to mount at /dev [CONFIG_DEVTMPFS]

Kernel hacking  --->
       Choose kernel unwinder (Frame pointer unwinder)  ---> [CONFIG_UNWINDER_FRAME_POINTE
```

시스템 요건에 따라 몇 가지 다른 옵션이 필요할 수도 있다. BLFS 패키지에 필요한 옵션 목록은 BLFS Index of Kernel Settings (http://www.linuxfromscratch.org/blfs/view/9.1/longindex.html#kernel-config-index)를 참고하라.

참고

호스트 컴퓨터가 UEFI를 사용 중이라면, 위의 'make defconfig'는 일부 EFI 관련 커널 옵션을 자동으로 추가할 것이다.

호스트의 UEFI 부팅 환경 내에서 LFS 커널을 부팅하려면 커널에 다음 옵션이 선택되어 있어야 한다:

```
Processor type and features  --->
  [*]   EFI stub support  [CONFIG_EFI_STUB]
```

LFS 내에서 UEFI 환경을 관리을 관리하는 법에 관한 더 자세한 설명은 http://www.linuxfromscratch.org/hints/downloads/files/lfs-uefi.txt의 lfs-uefi.txt에서 다룬다.

위 설정 항목에 대한 해석:

support for uevent helper
: Udev/Eudev을 사용할 때 이 옵션을 설정하면 장치 관리에 방해가 될 수 있다.

Maintain a devtmpfs
: 이렇게 하면 Udev 실행 없이도 커널에 의해 설정되는 자동화된 장치 노드가 생성된다. 그러면 Udev는 이 위에서 실행되어 권한을 관리하고 심볼릭 링크를 추가한다. 이 설정은 모든 Udev/Eudev 사용자에게 필요하다.

```
make menuconfig
```

선택적 make 환경 변수들의 의미:

LANG=<host_LANG_value> LC_ALL=
: 로케일을 호스트와 동일하게 설정한다. 이는 UTF-8 리눅스 텍스트 콘솔에서 적절한 menuconfig ncurses 인터페이스 행을 출력하는 데에 필요할 수 있다.
사용할 경우, 반드시 <host_LANG_value>를 호스트의 $LANG 변수의 값으로 대체하라. 또는 호스트의 $LC_ALL이나 $LC_CTYPE 값을 대신 사용할 수도 있다.

또 어떤 상황에서는 make oldconfig가 더 적절할 수 있다. 자세한 정보는 README 파일을 참고하라.

원한다면 커널 구성 파일인 .config를 호스트 시스템(해당 파일이 호스트에 존재한다고 가정한다)에서 압축 해제된 linux-5.5.3 디렉토리로 복사하여 커널 설정을 건너 뛰어도 좋다. 그러나 이 옵션은 권장하지는 않는다. 모든 설정 메뉴를 탐색하고 처음부터 커널 구성을 만드는 것이 더 좋다.

커널 이미지와 모듈들을 컴파일한다:

```
make
```

커널 모듈을 사용한다면, /etc/modprobe.d의 모듈 설정이 필요하다. 모듈 및 커널 설정에 관련된 정보는 7.3절. “장치 및 모듈 핸들링 개요”와 linux-5.5.3/Documentation 디렉토리의 커널 문서에 수록되어 있다. 또 modprobe.d(5)도 유용할 것이다.

커널 설정에서 모듈 지원이 비활성화되지 않았다면 다음과 같이 모듈을 설치하라:

```
make modules_install
```

커널 컴파일이 완료된 후, 설치를 완료하기 위해선 추가 절차가 필요하다. 일부 파일을 /boot 디렉토리에 복사해야 한다.

경고

호스트 시스템에 별도의 /boot 파티션이 있다면 아래 지침에서 복사할 파일을 그 곳에 넣어야 한다. 가장 쉬운 방법은 호스트(chroot 외부)에서 /boot를 /mnt/lfs/boot에 바인딩하는 것이다. 호스트 시스템에서 root 유저로 다음을 실행하라:

```
mount --bind /boot /mnt/lfs/boot
```

커널 이미지의 경로는 사용 중인 플랫폼에 따라 달라질 수 있다. 아래 파일 이름은 취향껏 변경할 수 있지만, 파일 이름의 어근은 다음 절에서 설명할 부팅 절차의 자동 설정과 호환되도록 vmlinuz여야 한다. 다음 명령은 x86 아키텍쳐를 가정한다:

```
cp -iv arch/x86/boot/bzImage /boot/vmlinuz-5.5.3-lfs-9.1
```

System.map은 커널을 위한 심볼 파일이다. 실행 중인 커널에 대한 커널 데이터 구조의 주소뿐만 아니라 커널 API에 있는 모든 함수의 엔트리 포인트를 매핑한다. 이는 커널 문제를 조사할 때 자료로 사용된다. 다음 명령을 실행하여 맵 파일을 설치하라:

```
cp -iv System.map /boot/System.map-5.5.3
```

위의 make menuconfig 단계에서 생성된 커널 설정 파일 .config에는 방금 컴파일된 커널에 대한 모든 설정 사항이 포함되어 있다. 나중에 참고할 수 있도록 이 파일을 보관하는 것이 좋다:

```
cp -iv .config /boot/config-5.5.3
```

리눅스 커널에 대한 문서를 설치한다:

```
install -d /usr/share/doc/linux-5.5.3
cp -r Documentation/* /usr/share/doc/linux-5.5.3
```

커널 소스 디렉토리의 파일은 root에 의해 소유되지 않는다는 점에 주의해야 한다. 패키지가 root 유저에 의해 압축 해제될 때는(우리가 chroot 안에서 그랬듯이), 그 파일들은 패키지를 압축한 사람의 컴퓨터에 있던 사용자 및 그룹 ID를 갖는다. 이는 다른 모든 패키지들에 대해선 설치 후 소스 트리가 제거되기 때문에 문제가 되지 않지만, 리눅스 소스 트리는 보통 오랫동안 보존한다. 때문에 패키징을 한 사용자의 ID 번호가 시스템의 누군가에게 할당될 가능성이 있다. 그러면 그 사람은 커널 소스에 대한 쓰기 권한을 갖게 된다.

참고

대부분의 경우 나중에 BLFS에서 설치될 패키지에 맞게 커널 구성을 업데이트해야 한다. 다른 패키지와 달리 새로 빌드된 커널을 설치한 후에는 커널 소스 트리를 제거할 필요가 없다.

커널 소스 트리를 보존하려면 linux-5.5.3 디렉토리에서 chown -R 0:0을 실행하여 모든 파일이 root 유저에 의해 소유되도록 하라.

주의

일부 커널 문서에서는 /usr/src/linux에 커널 소스 디렉토리를 가리키는 심볼릭 링크를 생성할 것을 권장한다. 이는 2.6 버전 이전의 커널에 한정되며, 기본 LFS 시스템이 완성되면 구축하고자 하는 패키지에 문제를 일으킬 수 있으므로 LFS 시스템에서 생성해서는 안 된다.

주의

시스템의 include 디렉토리(/usr/include)의 헤더들은 항상 Glibc가 컴파일된 헤더, 즉 6.7절. “리눅스-5.5.3 API 헤더”에서 설치된 수정 헤더(the sanitised headers)여야 한다. 따라서 원시 커널 헤더나 다른 커널 수정 헤더로 대체해서는 안 된다.

8.3.2. 리눅스 모듈 로드 순서 설정

리눅스 모듈은 대부분 자동으로 로드되지만, 때로는 특별한 방법이 필요할 때도 있다. modprobe나 insmod와 같이 모듈을 로드하는 프로그램들은 /etc/modprobe.d/usb.conf를 이러한 목적으로 사용한다. USB 드라이버(ehci_hcd, ohci_hcd 및 uhci_hcd)가 모듈로 빌드된 경우 올바른 순서로 로드되도록 하기 위해선 이 파일을 생성해야 한다; 부팅 시 경고 출력을 피하려면 ehci_hcd가 ohci_hcd와 uhci_hcd보다 먼저 로드되어야 한다.

다음을 실행하여 새 /etc/modprobe.d/usb.conf 파일을 생성하라:

```
install -v -m755 -d /etc/modprobe.d
cat > /etc/modprobe.d/usb.conf << "EOF"
# Begin /etc/modprobe.d/usb.conf

install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i uhci_hcd ; true

# End /etc/modprobe.d/usb.conf
EOF
```

8.3.3. 리눅스의 내용

설치된 파일들:	config-5.5.3, vmlinuz-5.5.3-lfs-9.1, System.map-5.5.3
설치된 디렉토리들:	/lib/modules, /usr/share/doc/linux-5.5.3

요약

config-5.5.3	커널에 대한 모든 구성 선택 사항 포함
vmlinuz-5.5.3-lfs-9.1	리눅스 시스템의 엔진. 컴퓨터를 켤 때, 로딩되는 운영 체제의 첫 번째 부분이 이 커널이다. 컴퓨터 하드웨어의 모든 요소를 감지하고 초기화한 다음, 이러한 부품들을 소프트웨어의 파일 트리로 사용할 수 있게 하며 단일 CPU로 하여금 겉보기에 동시에 여러 프로그램을 실행하는 멀티 태스킹이 가능하도록 한다.
System.map-5.5.3	주소 및 심볼 목록. 커널에 있는 모든 함수와 데이터 구조의 엔트리 포인트와 주소를 매핑한다.

8.4. GRUB을 사용하여 부팅 프로세스 설정

8.4.1. 도입

주의

GRUB을 잘못 구성하면 CD-ROM과 같은 대체 부팅 장치없이는 시스템이 작동하지 않게 될 수 있다. 이 절은 LFS 시스템을 부팅하는 데 필요하지 않다. Grub-Legacy, GRUB2, LILO와 같은 현재 사용 중인 부트 로더를 수정하는 편이 나을 수도 있다.

컴퓨터를 사용할 수 없게 되었을 때(부팅이 불가능한 경우)를 대비해 이를 “되살릴” 수 있도록 비상 부팅 디스크를 준비하라. 부팅 장치가 아직 없다면 생성하는 것이 좋다. 아래 절차를 제대로 따르려면 BLFS로 이동하여 libisoburn 패키지에서 xorriso를 설치해야 한다.

```
cd /tmp
grub-mkrescue --output=grub-img.iso
xorriso -as cdrecord -v dev=/dev/cdrw blank=as_needed grub-img.iso
```

참고

UEFI를 사용하도록 설정한 호스트 시스템에서 LFS를 부팅하려면 커널을 이전 절에서 설명한 CONFIG_EFI_STUB 기능을 사용하여 빌드해야 한다. 그러나 LFS는 그러한 추가없이 GRUB2를 사용하여 부팅할 수 있다. 이렇게 하려면 호스트 시스템 BIOS의 UEFI 모드 및 보안 부팅 기능을 비활성화 해야 한다. 자세한 내용은 http://www.linuxfromscratch.org/hints/downloads/files/lfs-uefi.txt에서 lfs-uefi.txt 힌트를 참고하라.

8.4.2. GRUB 명명 규칙

GRUB은 (hdn,m) 형식으로 드라이브 및 파티션에 고유한 명명 방식을 사용한다. 여기서 n은 하드 드라이브 번호이고 m은 파티션 번호이다. 하드 드라이브 번호는 0부터 시작하지만 파티션 번호는 일반 파티션의 경우 1부터, 확장 파티션의 경우 5부터 시작한다. 이는 두 숫자가 모두 0에서 시작하는 이전 버전과 다르다는 점에 주의하라. 예를 들어, 파티션 sda1은 GRUB에서 (hd0,1)이고 sdb3은 (hd1,3)이다. 리눅스와 달리 GRUB은 CD-ROM 드라이브를 하드 드라이브로 간주하지 않는다. 예를 들어, hdb에서 CD를 사용하고, hdc에서 두 번째 하드 드라이브를 사용한다면, 두 번째 하드 드라이브는 여전히 (hd1)이다.

8.4.3. 구성 설정

GRUB은 하드 디스크의 물리적 첫 번째 트랙에 데이터를 쓰는 방식으로 작동한다. 이 영역은 파일 시스템의 일부가 아니다. 그 곳의 프로그램은 부트 파티션의 GRUB 모듈에 액세스한다. 기본 위치는 /boot/grub/이다.

부트 파티션의 위치는 구성하는 사용자의 선택에 달렸다. 한 가지 권장 사항은 부팅 정보를 위한 별도의 작은(권장 크기는 100 MB) 파티션을 만드는 것이다. 이렇게 하면 LFS 또는 일부 상용 배포판에 관계없이 각 빌드가 동일한 부팅 파일에 액세스할 수 있으며 부팅된 모든 시스템에서 액세스할 수 있다. 이렇게 하려면 별도의 파티션을 마운트하고 현재 /boot 디렉토리의 모든 파일(가령 이전 절에서 방금 빌드한 리눅스 커널)을 새 파티션으로 이동해야 한다. 그런 다음 파티션을 마운트 해제하고 /boot로 다시 마운트하라. 이렇게 할 경우 반드시 /etc/fstab을 업데이트하라.

현재의 lfs 파티션을 사용하는 것도 문제없지만 여러 시스템에 대한 구성은 더 까다롭다.

위의 정보를 사용하여 root 파티션(또는 별도의 파티션을 사용한다면 부트 파티션)의 적절한 이름를 확인하라. 다음 예시에서는 root 파티션(또는 별도의 부트 파티션)이 sda2인 것으로 가정한다.

GRUB 파일들을 /boot/grub에 설치하고 부트 경로를 설정한다:

주의

다음 명령은 현재 부트 로더를 덮어쓴다. 다른 부팅 관리자로 마스터 부트 레코드(Master Boot Record, MBR)를 관리하는 경우와 같이, 덮어쓰기를 원하지 않는다면 이 명령을 실행하지 않도록 하라.

```
grub-install /dev/sda
```

참고

시스템이 UEFI를 사용하여 부팅된 경우 grub-install은 x86_64-efi를 대상으로 한 설치를 시도할 것이다. 그러나 해당 파일들은 6장에서 설치되지 않았다. 이 경우 위의 명령에 --target i386-pc를 추가하라.

8.4.4. GRUB 구성 파일 생성

/boot/grub/grub.cfg를 생성하라:

```
cat > /boot/grub/grub.cfg << "EOF"
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5

insmod ext2
set root=(hd0,2)

menuentry "GNU/Linux, Linux 5.5.3-lfs-9.1" {
        linux   /boot/vmlinuz-5.5.3-lfs-9.1 root=/dev/sda2 ro
}
EOF
```

참고

GRUB의 관점에서 커널 파일은 사용된 파티션에 상대적이다. 별도의 /boot 파티션을 사용한다면 위의 linux 행에서 /boot를 제거하라. 또 set root 행이 부트 파티션을 가리키도록 수정해야 한다.

GRUB은 매우 강력한 프로그램으로 다양한 장치, 운영 체제 및 파티션 유형에서 부팅할 수 있는 엄청난 수의 옵션을 제공한다. 더불어 그래픽 스플래시 화면, 소리 재생, 마우스 입력 등과 같은 사용자 정의를 위한 많은 옵션들도 있다. 이러한 옵션에 대한 자세한 내용들은 본 주제의 범위를 벗어난다.

경고

구성 파일을 자동으로 작성할 수 있는 grub-mkconfig라는 명령이 있다. 이것은 /etc/grub.d/의 스크립트들을 사용하는데, 사용자가 만드는 모든 사용자 정의를 파괴한다. 이 스크립트들은 주로 소스가 없는 배포판용(for non-source distributions)으로 설계되었으며 LFS에는 권장되지 않는다. 만약 상용 리눅스 배포판을 설치한다면 이 프로그램이 실행될 가능성이 농후하다. 반드시 grub.cfg 파일을 백업하라.

9장. 결론

9.1. 결론

축하한다! 새로운 LFS 시스템이 설치되었다! 우리는 당신의 빛나는 새 맞춤 제작 리눅스 시스템으로 많은 성공을 거두길 기원한다.

/etc/lfs-release 파일을 생성하는 것이 좋다. 이 파일이 있으면 시스템에 어떤 LFS 버전이 설치되어 있는지 쉽게 확인할 수 있다(또 언젠가 당신에게 도움이 필요할 때 우리에게도 유용하다). 다음을 실행하여 이 파일을 생성하라:

```
echo 9.1 > /etc/lfs-release
```

설치된 시스템을 설명하는 두 개의 파일을 바이너리 형식으로 또는 빌드하여, 시스템에 설치될 패키지에서 사용할 수 있다.

첫 번째는 Linux Standards Base(LSB)와 관련된 새 시스템의 상태를 보여준다. 이 파일을 작성하려면 다음을 실행하라:

```
cat > /etc/lsb-release << "EOF"
DISTRIB_ID="Linux From Scratch"
DISTRIB_RELEASE="9.1"
DISTRIB_CODENAME="<your name here>"
DISTRIB_DESCRIPTION="Linux From Scratch"
EOF
```

두 번째 것은 거의 동일한 정보를 담고 있으며, systemd와 일부 그래픽 데스크톱 환경에서 사용된다. 이 파일을 만드려면 실행하라:

```
cat > /etc/os-release << "EOF"
NAME="Linux From Scratch"
VERSION="9.1"
ID=lfs
PRETTY_NAME="Linux From Scratch 9.1"
VERSION_CODENAME="<your name here>"
EOF
```

시스템을 당신 고유의 것으로 만드려면 'DISTRIB_CODENAME'과 'VERSION_CODENAME' 필드를 취향껏 작성하라.

9.2. 통계 등록

이제 이 책을 모두 끝마쳤으니 LFS 사용자로 등록하는 것이 어떤가? http://www.linuxfromscratch.org/cgi-bin/lfscounter.php 이 곳으로 이동하여 사용자 이름과 사용했던 첫 번째 LFS 버전을 입력해서 LFS 사용자로 등록하라.

이제 LFS로 재부팅해보자.

9.3. 시스템 재부팅

이제 모든 소프트웨어가 설치되었으니 컴퓨터를 다시 부팅할 시간이다. 그러나 몇 가지 알아야 할 것이 있다. 이 책에서 만든 시스템은 아주 최소한의 것이며 앞으로 계속 사용하는 데 필요한 기능이 없을 가능성이 크다. 현재 chroot 환경에 있는 동안 BLFS 책에서 몇 가지 추가 패키지를 설치하면 새 LFS 시스템으로 재부팅한 후에도 계속 사용할 수 있는 훨씬 더 좋은 환경을 만들 수 있다. 다음은 몇 가지 제안 사항이다:

- Lynx와 같은 텍스트 모드 브라우저를 사용하면 한 가상 터미널에서 BLFS 책을 쉽게 보면서 다른 가상 터미널에서 패키지를 빌드할 수 있다.
- GPM 패키지를 통해 가상 터미널에서 복사/붙여넣기 작업을 수행할 수 있다.
- 정적 IP 구성이 네트워킹 요구 사항을 충족하지 못하는 상황이라면, dhcpcd나 dhcp의 클라이언트 부분을 설치하면 도움이 될 것이다.
- sudo를 설치하면 root 사용자가 아닌 사용자로써 패키지를 빌드하고 새 시스템에 결과물을 쉽게 설치하는 데 유용하다.
- 편안한 GUI 환경의 원격 시스템에서 새 시스템에 접속하려면 openssh를 설치하라.
- 인터넷을 통해 파일을 쉽게 받으려면 wget를 설치하라.
- 하나 이상의 디스크 드라이브에 GUID 파티션 테이블(GPT)이 있다면 gptfdisk나 parted가 유용할 것이다.
- 마지막으로, 이 시점에서는 다음과 같은 설정 파일들을 검토하는 것도 좋다.
  - /etc/bashrc
  - /etc/dircolors
  - /etc/fstab
  - /etc/hosts
  - /etc/inputrc
  - /etc/profile
  - /etc/resolv.conf
  - /etc/vimrc
  - /root/.bash_profile
  - /root/.bashrc
  - /etc/sysconfig/ifconfig.eth0

이제 충분한 것 같으니, 멋진 새 LFS 시스템을 처음으로 부팅해보자! 우선 chroot 환경에서 빠져나간다:

```
logout
```

그리고 가상 파일 시스템들을 마운트 해제한다:

```
umount -v $LFS/dev/pts
umount -v $LFS/dev
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
```

LFS 파일 시스템 자체를 마운트 해제한다:

```
umount -v $LFS
```

여러 파티션이 생성된 경우 다음과 같이 기본 파티션을 마운트 해제하기 전에 다른 파티션을 먼저 마운트 해제하라:

```
umount -v $LFS/usr
umount -v $LFS/home
umount -v $LFS
```

이제, 시스템을 재부팅한다:

```
shutdown -r now
```

앞서 설명한 대로 GRUB 부트 로더가 설정되었다고 가정하면 메뉴는 LFS 9.1을 자동으로 부팅하도록 설정된다.

재부팅이 완료되면 LFS 시스템을 사용할 수 있으며 사용자의 필요에 맞게 소프트웨어를 추가할 수 있다.

9.4. 이젠 뭘 할까?

이 LFS 책을 읽어준 독자에게 감사드린다. 이 책이 도움이 되었기를, 시스템 구축 과정에 대해 더 많이 배웠기를 바란다.

이제 LFS 시스템이 설치되었으니, 여러분은 “이 다음엔 뭘 해야하나?”하고 궁금해할 것이다. 그 질문에 답하기 위해, 우리는 당신에게 도움이 될만한 리스트를 작성했다.

- 유지 보수

  버그와 보안 문제는 모든 소프트웨어에서 자주 보고된다. LFS 시스템은 소스에서 컴파일되므로, 이러한 보고들을 따라 관리하는 것은 당신에게 달려 있다. 이러한 보고서를 추적하는 몇 가지 온라인 출처가 있으며, 그 중 일부를 아래에 소개한다:

  - CERT (Computer Emergency Response Team)

    CERT에는 다양한 운영 체제 및 응용 프로그램에 대한 보안 경고를 게시하는 메일링 리스트가 있다. 구독 정보는 http://www.us-cert.gov/cas/signup.html에서 확인할 수 있다.

  - Bugtraq

    Bugtraq은 컴퓨터 보안에 관한 완전한 공개 메일링 리스트이다. 이 곳은 새롭게 발견된 보안 문제를 게시하고 가끔씩 이를 해결할 수 있는 방법들을 발표한다. 구독 정보는 http://www.securityfocus.com/archive에서 확인할 수 있다.

- Beyond Linux From Scratch

  The Beyond Linux From Scratch 책은 LFS 책의 범위를 넘어 광범위한 소프트웨어 설치 절차를 다룬다. 이 BLFS 프로젝트는 http://www.linuxfromscratch.org/blfs/에 있다.

- LFS Hints

  LFS 힌트는 LFS 커뮤니티의 자원봉사자들이 작성한 교육 문서 모음이다. 이 사이트는 http://www.linuxfromscratch.org/hints/list.html에서 이용할 수 있다.

- Mailing lists

  도움이 필요하거나, 최신 개발 버전으로 유지하고 싶거나, 프로젝트에 공헌하고 싶다면 구독할 수 있는 여러 LFS 메일링 리스트가 있다. 자세한 내용은 Chapter 1 - Mailing Lists을 참고하라.

- The Linux Documentation Project

  The Linux Documentation Project (TLDP)의 목표는 리눅스 문서의 모든 문제에 대해 협업하는 것이다. TLDP엔 많은 HOWTO, 가이드, man 페이지가 있다. http://www.tldp.org/에서 찾을 수 있다.

부 IV. 부록

부록 A. 줄임말과 용어

ABI	Application Binary Interface
ALFS	Automated Linux From Scratch
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BIOS	Basic Input/Output System
BLFS	Beyond Linux From Scratch
BSD	Berkeley Software Distribution
chroot	change root
CMOS	Complementary Metal Oxide Semiconductor
COS	Class Of Service
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CVS	Concurrent Versions System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
EGA	Enhanced Graphics Adapter
ELF	Executable and Linkable Format
EOF	End of File
EQN	equation
ext2	second extended file system
ext3	third extended file system
ext4	fourth extended file system
FAQ	Frequently Asked Questions
FHS	Filesystem Hierarchy Standard
FIFO	First-In, First Out
FQDN	Fully Qualified Domain Name
FTP	File Transfer Protocol
GB	Gigabytes
GCC	GNU Compiler Collection
GID	Group Identifier
GMT	Greenwich Mean Time
HTML	Hypertext Markup Language
IDE	Integrated Drive Electronics
IEEE	Institute of Electrical and Electronic Engineers

IO	Input/Output
IP	Internet Protocol
IPC	Inter-Process Communication
IRC	Internet Relay Chat
ISO	International Organization for Standardization
ISP	Internet Service Provider
KB	Kilobytes
LED	Light Emitting Diode
LFS	Linux From Scratch
LSB	Linux Standard Base
MB	Megabytes
MBR	Master Boot Record
MD5	Message Digest 5
NIC	Network Interface Card
NLS	Native Language Support
NNTP	Network News Transport Protocol
NPTL	Native POSIX Threading Library
OSS	Open Sound System
PCH	Pre-Compiled Headers
PCRE	Perl Compatible Regular Expression
PID	Process Identifier
PTY	pseudo terminal
QOS	Quality Of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
RTC	Real Time Clock
SBU	Standard Build Unit
SCO	The Santa Cruz Operation
SHA1	Secure-Hash Algorithm 1
TLDP	The Linux Documentation Project
TFTP	Trivial File Transfer Protocol
TLS	Thread-Local Storage
UID	User Identifier
umask	user file-creation mask
USB	Universal Serial Bus
UTC	Coordinated Universal Time

UUID	Universally Unique Identifier
VC	Virtual Console
VGA	Video Graphics Array
VT	Virtual Terminal

부록 B. 감사의 말

우리는 Linux From Scratch 프로젝트에 기여한 다음의 사람들과 단체들에게 감사를 표하고 싶다.

- Gerard Beekmans <gerard@linuxfromscratch.org> - LFS 창립자
- Bruce Dubbs <bdubbs@linuxfromscratch.org> - LFS 관리 편집자
- Jim Gifford <jim@linuxfromscratch.org> - CLFS 프로젝트 공동 리더
- Pierre Labastie <pierre@linuxfromscratch.org> - BLFS 편집자 겸 ALFS 리더
- DJ Lucas <dj@linuxfromscratch.org> - LFS 및 BLFS 편집자
- Ken Moffat <ken@linuxfromscratch.org> - BLFS 편집자
- 다양한 LFS와 BLFS 메일링 리스트에 있는 수많은 다른 사람들이 그들의 제안, 책 테스트, 버그 리포트, 실험, 그리고 다양한 패키지 설치 경험들을 기부함으로써 이 책을 가능하게 했다.

번역자

- Manuel Canales Esparcia <macana@macana-es.com> - 스페인어 LFS 번역 프로젝트
- Johan Lenglet <johan@linuxfromscratch.org> - 프랑스어 LFS 번역 프로젝트, 2008년까지
- Jean-Philippe Mengual <jmengual@linuxfromscratch.org> - 프랑스어 LFS 번역 프로젝트 2008-2016
- Julien Lepiller <jlepiller@linuxfromscratch.org> - 프랑스어 LFS 번역 프로젝트 2017-현재
- Anderson Lizardo <lizardo@linuxfromscratch.org> - 포르투갈어 LFS 번역 프로젝트
- Thomas Reitelbach <tr@erdfunkstelle.de> - 독일어 LFS 번역 프로젝트
- Anton Maisak <info@linuxfromscratch.org.ru> - 러시아어 LFS 번역 프로젝트
- Elena Shevcova <helen@linuxfromscratch.org.ru> - 러시아어 LFS 번역 프로젝트

미러 관리자들

북미 미러

- Scott Kveton <scott@osuosl.org> - lfs.oregonstate.edu mirror
- William Astle <lost@l-w.net> - ca.linuxfromscratch.org mirror
- Eujon Sellers <jpolen@rackspace.com> - lfs.introspeed.com mirror
- Justin Knierim <tim@idge.net> - lfs-matrix.net mirror

남미 미러

- Manuel Canales Esparcia <manuel@linuxfromscratch.org> - lfsmirror.lfs-es.info mirror
- Luis Falcon <Luis Falcon> - torredehanoi.org mirror

유럽 미러

- Guido Passet <guido@primerelay.net> - nl.linuxfromscratch.org mirror
- Bastiaan Jacques <baafie@planet.nl> - lfs.pagefault.net mirror
- Sven Cranshoff <sven.cranshoff@lineo.be> - lfs.lineo.be mirror

- Scarlet Belgium - lfs.scarlet.be mirror
- Sebastian Faulborn <info@aliensoft.org> - lfs.aliensoft.org mirror
- Stuart Fox <stuart@dontuse.ms> - lfs.dontuse.ms mirror
- Ralf Uhlemann <admin@realhost.de> - lfs.oss-mirror.org mirror
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> - at.linuxfromscratch.org mirror
- Fredrik Danerklint <fredan-lfs@fredan.org> - se.linuxfromscratch.org mirror
- Franck <franck@linuxpourtous.com> - lfs.linuxpourtous.com mirror
- Philippe Baque <baque@cict.fr> - lfs.cict.fr mirror
- Vitaly Chekasin <gyouja@pilgrims.ru> - lfs.pilgrims.ru mirror
- Benjamin Heil <kontakt@wankoo.org> - lfs.wankoo.org mirror
- Anton Maisak <info@linuxfromscratch.org.ru> - linuxfromscratch.org.ru mirror

아시아 미러

- Satit Phermsawang <satit@wbac.ac.th> - lfs.phayoune.org mirror
- Shizunet Co.,Ltd. <info@shizu-net.jp> - lfs.mirror.shizu-net.jp mirror
- Init World <http://www.initworld.com/> - lfs.initworld.com mirror

호주 미러

- Jason Andrade <jason@dstc.edu.au> - au.linuxfromscratch.org mirror

이전 프로젝트 팀 구성원

- Christine Barczak <theladyskye@linuxfromscratch.org> - LFS 책 편집자
- Archaic <archaic@linuxfromscratch.org> - LFS 기술 작가/편집자, HLFS 프로젝트 리더, BLFS 편집자, Hints and Patches 프로젝트 관리자
- Matthew Burgess <matthew@linuxfromscratch.org> - LFS 프로젝트 리더, LFS 기술 작가/편집자
- Nathan Coulson <nathan@linuxfromscratch.org> - LFS-부트스크립트 관리자
- Timothy Bauscher
- Robert Briggs
- Ian Chilton
- Jeroen Coumans <jeroen@linuxfromscratch.org> - 웹사이트 개발자, FAQ 관리자
- Manuel Canales Esparcia <manuel@linuxfromscratch.org> - LFS/BLFS/HLFS XML 및 XSL 관리자
- Alex Groenewoud - LFS 기술 작가
- Marc Heerdink
- Jeremy Huntwork <jhuntwork@linuxfromscratch.org> - LFS 기술 작가, LFS LiveCD 관리자
- Bryan Kadzban <bryan@linuxfromscratch.org> - LFS 기술 작가
- Mark Hymers
- Seth W. Klein - FAQ 관리자

- Nicholas Leippe <nicholas@linuxfromscratch.org> - 위키 관리자
- Anderson Lizardo <lizardo@linuxfromscratch.org> - 웹사이트 백엔드-스크립트 관리자
- Randy McMurchy <randy@linuxfromscratch.org> - BLFS 프로젝트 리더, LFS 편집자
- Dan Nicholson <dnicholson@linuxfromscratch.org> - LFS 및 BLFS 편집자
- Alexander E. Patrakov <alexander@linuxfromscratch.org> - LFS 기술 작가, LFS 다국어화 작가, LFS Live CD 관리자
- Simon Perreault
- Scot Mc Pherson <scot@linuxfromscratch.org> - LFS NNTP 게이트웨이 관리자
- Douglas R. Reno <renodr@linuxfromscratch.org> - Systemd 편집자
- Ryan Oliver <ryan@linuxfromscratch.org> - CLFS 프로젝트 공동 리더
- Greg Schafer <gschafer@zip.com.au> - LFS 기술 작가 겸 차세대 64비트 지원 빌드 방법 설계자
- Jesse Tie-Ten-Quee - LFS 기술 작가
- James Robertson <jwrober@linuxfromscratch.org> - Bugzilla 관리자
- Tushar Teredesai <tushar@linuxfromscratch.org> - BLFS 책 편집자, Hints and Patches 프로젝트 리더
- Jeremy Utley <jeremy@linuxfromscratch.org> - LFS 기술 작가, Bugzilla 관리자, LFS-부트스크립트 관리자
- Zack Winkles <zwinkles@gmail.com> - LFS 기술 작가

부록 C. 의존성

LFS에 빌드된 모든 패키지는 올바르게 빌드 및 설치되기 위해 하나 이상의 다른 패키지에 의존한다. 어떤 패키지는 순환 의존성에도 속한다. 즉, 첫 번째 패키지는 두 번째 패키지에 의존하고, 그 다음 패키지는 첫 번째 패키지에 의존한다. 이러한 의존성 때문에 LFS에 패키지가 빌드되는 순서는 매우 중요하다. 이 페이지의 목적은 LFS에 내장된 각 패키지의 의존성을 문서화하는 것이다.

우리가 만드는 각각의 패키지에 대해 우리는 세 가지, 때로는 네 가지 유형의 의존성을 명시했다. 첫 번째 목록에는 해당 패키지를 컴파일하고 설치하기 위해 필요한 다른 패키지가 명시되어 있다. 두 번째는 첫 번째 목록에 있는 패키지와 더불어, 테스트 스위트를 실행하기 위해 필요한 패키지가 명시되어 있다. 세 번째 의존성 목록은 빌드하고 설치되기 전에 이 패키지를 최종 위치에 설치해야 하는 패키지들이다. 이는 대부분 그 패키지들의 스크립트 내에서 바이너리 파일에 대한 경로를 하드 코드화하기 때문이다. 특정 순서로 빌드되지 않으면 /tools/bin/[binary] 경로가 최종 시스템에 설치된 스크립트 내부에 위치할 수 있다. 이는 분명히 바람직하지 않다.

마지막 의존성 목록은 LFS에서 다루지 않지만 사용자에게 유용할 수 있는 선택적 패키지다. 이 패키지들에는 자체적으로 필수 또는 선택적 의존성이 추가로 있을 수 있다. 이러한 의존성 패키지들은 LFS 책을 완료한 후 설치한 다음, 앞 내용으로 돌아가서 LFS 패키지를 다시 빌드하는 것이 권장되는 방법이다. 몇 가지는 재설치를 BLFS에서 다룬다.

Acl

먼저 설치되어야 하는 패키지:	Attr, Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, Texinfo
테스트 스위트에 필요한 패키지:	Automake, Diffutils, Findutils, Libtool
이 패키지를 필요로 하는 패키지:	Coreutils, Sed, Tar, Vim
선택적 의존성 패키지:	없음

Attr

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Gettext, Grep, M4, Make, Perl, Sed, Texinfo
테스트 스위트에 필요한 패키지:	Automake, Diffutils, Findutils, Libtool
이 패키지를 필요로 하는 패키지:	Acl, Libcap
선택적 의존성 패키지:	없음

Autoconf

먼저 설치되어야 하는 패키지:	Bash, Coreutils, Grep, M4, Make, Perl, Sed, Texinfo
테스트 스위트에 필요한 패키지:	Automake, Diffutils, Findutils, GCC, Libtool
이 패키지를 필요로 하는 패키지:	Automake
선택적 의존성 패키지:	Emacs

Automake

먼저 설치되어야 하는 패키지: Autoconf, Bash, Coreutils, Gettext, Grep, M4, Make, Perl, Sed, Texinfo
테스트 스위트에 필요한 패키지: Binutils, Bison, Bzip2, DejaGNU, Diffutils, Expect, Findutils, Flex, GCC, Gettext, Gzip, Libtool, Tar
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Bash

먼저 설치되어야 하는 패키지: Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Readline, Sed, Texinfo
테스트 스위트에 필요한 패키지: Shadow
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: Xorg

Bc

먼저 설치되어야 하는 패키지: Bash, Binutils, Bison, Coreutils, GCC, Glibc, Grep, Make, Perl, Readline
테스트 스위트에 필요한 패키지: Gawk
이 패키지를 필요로 하는 패키지: Linux Kernel
선택적 의존성 패키지: 없음

Binutils

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, File, Gawk, GCC, Glibc, Grep, Make, Perl, Sed, Texinfo, Zlib
테스트 스위트에 필요한 패키지: DejaGNU, Expect
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Bison

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Perl, Sed
테스트 스위트에 필요한 패키지: Diffutils, Findutils, Flex
이 패키지를 필요로 하는 패키지: Kbd, Tar
선택적 의존성 패키지: Doxygen (test suite)

Bzip2

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make, Patch

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: File

선택적 의존성 패키지: 없음

Check

먼저 설치되어야 하는 패키지: GCC, Grep, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Coreutils

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Libcap, Make, Patch, Perl, Sed, Texinfo

테스트 스위트에 필요한 패키지: Diffutils, E2fsprogs, Findutils, Shadow, Util-linux

이 패키지를 필요로 하는 패키지: Bash, Diffutils, Eudev, Findutils, Man-DB

선택적 의존성 패키지: Perl Expect, IO:Tty modules (for test suite)

DejaGNU

먼저 설치되어야 하는 패키지: Bash, Coreutils, Diffutils, GCC, Grep, Make, Sed

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Diffutils

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: Perl

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

E2fsprogs

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Gzip, Make, Sed, Texinfo, Util-linux

테스트 스위트에 필요한 패키지: Procps-ng, Psmisc

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Eudev

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Gperf, Make, Sed

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Expat

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: XML::Parser

선택적 의존성 패키지: 없음

Expect

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Patch, Sed, Tcl

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

File

먼저 설치되어야 하는 패키지: Bash, Binutils, Bzip2, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Xz, Zlib

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Findutils

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: DejaGNU, Diffutils, Expect

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Flex

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, M4, Make, Patch, Sed, Texinfo

테스트 스위트에 필요한 패키지: Bison, Gawk

이 패키지를 필요로 하는 패키지: IPRoute2, Kbd, Man-DB

선택적 의존성 패키지: 없음

Gawk

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, GMP, Grep, Make, MPFR, Patch, Readline, Sed, Texinfo

테스트 스위트에 필요한 패키지: Diffutils

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Gcc

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, GMP, Grep, M4, Make, MPC, MPFR, Patch, Perl, Sed, Tar, Texinfo

테스트 스위트에 필요한 패키지: DejaGNU, Expect, Shadow

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: GNAT, ISL

GDBM

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Grep, Make, Sed

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Gettext

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지: Diffutils, Perl, Tcl
이 패키지를 필요로 하는 패키지: Automake
선택적 의존성 패키지: 없음

Glibc

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Gzip, Linux API Headers, Make, Perl, Python, Sed, Texinfo
테스트 스위트에 필요한 패키지: File
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

GMP

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, M4, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: MPFR, GCC
선택적 의존성 패키지: 없음

Gperf

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Glibc, Make
테스트 스위트에 필요한 패키지: Diffutils, Expect
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Grep

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Patch, Sed, Texinfo
테스트 스위트에 필요한 패키지: Gawk
이 패키지를 필요로 하는 패키지: Man-DB
선택적 의존성 패키지: Pcre

Groff

먼저 설치되어야 하는 패키지:	Bash, Binutils, Bison, Coreutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed, Texinfo
테스트 스위트에 필요한 패키지:	테스트 스위트 없음
이 패키지를 필요로 하는 패키지:	Man-DB, Perl
선택적 의존성 패키지:	Ghostscript

GRUB

먼저 설치되어야 하는 패키지:	Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Texinfo, Xz
테스트 스위트에 필요한 패키지:	없음
이 패키지를 필요로 하는 패키지:	없음
선택적 의존성 패키지:	없음

Gzip

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지:	Diffutils, Less
이 패키지를 필요로 하는 패키지:	Man-DB
선택적 의존성 패키지:	없음

Iana-Etc

먼저 설치되어야 하는 패키지:	Coreutils, Gawk, Make
테스트 스위트에 필요한 패키지:	테스트 스위트 없음
이 패키지를 필요로 하는 패키지:	Perl
선택적 의존성 패키지:	없음

Inetutils

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo, Zlib
테스트 스위트에 필요한 패키지:	테스트 스위트 없음
이 패키지를 필요로 하는 패키지:	Tar
선택적 의존성 패키지:	없음

Intltool

먼저 설치되어야 하는 패키지: Bash, Gawk, Glibc, Make, Perl, Sed, XML::Parser
테스트 스위트에 필요한 패키지: Perl
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

IProute2

먼저 설치되어야 하는 패키지: Bash, Bison, Coreutils, Flex, GCC, Glibc, Make, Linux API Headers
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Kbd

먼저 설치되어야 하는 패키지: Bash, Binutils, Bison, Check, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Patch, Sed
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Kmod

먼저 설치되어야 하는 패키지: Bash, Binutils, Bison, Coreutils, Flex, GCC, Gettext, Glibc, Gzip, Make, Pkg-config, Sed, Xz-Utils, Zlib
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: Eudev
선택적 의존성 패키지: 없음

Less

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: Gzip
선택적 의존성 패키지: Pcre

Libcap

먼저 설치되어야 하는 패키지: Attr, Bash, Binutils, Coreutils, GCC, Glibc, Perl, Make, Sed

테스트 스위트에 필요한 패키지: 테스트 스위트 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: Linux-PAM

Libelf

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Glibc, Make

테스트 스위트에 필요한 패키지: 테스트 스위트 없음

이 패키지를 필요로 하는 패키지: Linux Kernel

선택적 의존성 패키지: 없음

Libffi

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Glibc, Make, Sed

테스트 스위트에 필요한 패키지: DejaGnu

이 패키지를 필요로 하는 패키지: Python

선택적 의존성 패키지: 없음

Libpipeline

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: Check

이 패키지를 필요로 하는 패키지: Man-DB

선택적 의존성 패키지: 없음

Libtool

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: Autoconf, Automake, Findutils

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Linux Kernel

먼저 설치되어야 하는 패키지: Bash, Bc, Binutils, Coreutils, Diffutils, Findutils, GCC, Glibc, Grep, Gzip, Kmod, Libelf, Make, Ncurses, OpenSSL, Perl, Sed
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

M4

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지: Diffutils
이 패키지를 필요로 하는 패키지: Autoconf, Bison
선택적 의존성 패키지: libsigsegv

Make

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지: Perl, Procps-ng
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Man-DB

먼저 설치되어야 하는 패키지: Bash, Binutils, Bzip2, Coreutils, Flex, GCC, GDBM, Gettext, Glibc, Grep, Groff, Gzip, Less, Libpipeline, Make, Sed, Xz
테스트 스위트에 필요한 패키지: Util-linux
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Man-Pages

먼저 설치되어야 하는 패키지: Bash, Coreutils, Make
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Meson

먼저 설치되어야 하는 패키지: Ninja, Python
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: Systemd
선택적 의존성 패키지: 없음

MPC

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, MPFR, Sed, Texinfo
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: GCC
선택적 의존성 패키지: 없음

MPFR

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, GMP, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: Gawk, GCC
선택적 의존성 패키지: 없음

Ncurses

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Patch, Sed
테스트 스위트에 필요한 패키지: 테스트 스위트 없음
이 패키지를 필요로 하는 패키지: Bash, GRUB, Inetutils, Less, Procps-ng, Psmisc, Readline, Texinfo, Util-linux, Vim
선택적 의존성 패키지: 없음

Ninja

먼저 설치되어야 하는 패키지: Binutils, Coreutils, Gcc, Python
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: Meson
선택적 의존성 패키지: Asciidoc, Doxygen, Emacs, re2c

Openssl

먼저 설치되어야 하는 패키지: Binutils, Coreutils, Gcc, Make, Perl
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: Linux
선택적 의존성 패키지: 없음

Patch

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Glibc, Grep, Make, Sed
테스트 스위트에 필요한 패키지: Diffutils
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: Ed

Perl

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, GDBM, Glibc, Grep, Groff, Make, Sed, Zlib
테스트 스위트에 필요한 패키지: Iana-Etc, Procps-ng
이 패키지를 필요로 하는 패키지: Autoconf
선택적 의존성 패키지: 없음

Pkg-config

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Popt, Sed
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: Kmod
선택적 의존성 패키지: 없음

Popt

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make
테스트 스위트에 필요한 패키지: Diffutils, Sed
이 패키지를 필요로 하는 패키지: Pkg-config
선택적 의존성 패키지: 없음

Procps-ng

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses
테스트 스위트에 필요한 패키지:	DejaGNU
이 패키지를 필요로 하는 패키지:	없음
선택적 의존성 패키지:	없음

Psmisc

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed
테스트 스위트에 필요한 패키지:	테스트 스위트 없음
이 패키지를 필요로 하는 패키지:	없음
선택적 의존성 패키지:	없음

Python

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Gdbm, Gettext, Glibc, Grep, Libffi, Make, Ncurses, Sed
테스트 스위트에 필요한 패키지:	GDB, Valgrind
이 패키지를 필요로 하는 패키지:	Ninja
선택적 의존성 패키지:	Berkeley DB, OpenSSL, SQLite, Tk

Readline

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Patch, Sed, Texinfo
테스트 스위트에 필요한 패키지:	테스트 스위트 없음
이 패키지를 필요로 하는 패키지:	Bash, Gawk
선택적 의존성 패키지:	없음

Sed

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo
테스트 스위트에 필요한 패키지:	Diffutils, Gawk
이 패키지를 필요로 하는 패키지:	E2fsprogs, File, Libtool, Shadow
선택적 의존성 패키지:	없음

Shadow

먼저 설치되어야 하는 패키지: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed

테스트 스위트에 필요한 패키지: 테스트 스위트 없음

이 패키지를 필요로 하는 패키지: Coreutils

선택적 의존성 패키지: Cracklib, PAM

Sysklogd

먼저 설치되어야 하는 패키지: Binutils, Coreutils, GCC, Glibc, Make, Patch

테스트 스위트에 필요한 패키지: 테스트 스위트 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Systemd

먼저 설치되어야 하는 패키지: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, Expat, Gawk, GCC, Glibc, Gperf, Grep, Intltool, Libcap, Meson, Sed, Util-linux

테스트 스위트에 필요한 패키지: 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: Many, see BLFS systemd page

Sysvinit

먼저 설치되어야 하는 패키지: Binutils, Coreutils, GCC, Glibc, Make, Sed

테스트 스위트에 필요한 패키지: 테스트 스위트 없음

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Tar

먼저 설치되어야 하는 패키지: Acl, Attr, Bash, Binutils, Bison, Coreutils, GCC, Gettext, Glibc, Grep, Inetutils, Make, Sed, Texinfo

테스트 스위트에 필요한 패키지: Autoconf, Diffutils, Findutils, Gawk, Gzip

이 패키지를 필요로 하는 패키지: 없음

선택적 의존성 패키지: 없음

Tcl

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Texinfo

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Patch, Sed
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: 없음

Util-linux

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Diffutils, Eudev, Findutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: Libcap-ng

Vim

먼저 설치되어야 하는 패키지: Acl, Attr, Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed
테스트 스위트에 필요한 패키지: 없음
이 패키지를 필요로 하는 패키지: 없음
선택적 의존성 패키지: Xorg, GTK+2, LessTif, Python, Tcl, Ruby, GPM

XML::Parser

먼저 설치되어야 하는 패키지: Bash, Binutils, Coreutils, Expat, GCC, Glibc, Make, Perl
테스트 스위트에 필요한 패키지: Perl
이 패키지를 필요로 하는 패키지: Intltool
선택적 의존성 패키지: 없음

Xz

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make
테스트 스위트에 필요한 패키지:	없음
이 패키지를 필요로 하는 패키지:	Eudev, File, GRUB, Kmod, Man-DB
선택적 의존성 패키지:	없음

Zlib

먼저 설치되어야 하는 패키지:	Bash, Binutils, Coreutils, GCC, Glibc, Make, Sed
테스트 스위트에 필요한 패키지:	없음
이 패키지를 필요로 하는 패키지:	File, Kmod, Perl, Util-linux
선택적 의존성 패키지:	없음

Zstd

먼저 설치되어야 하는 패키지:	Binutils, Coreutils, GCC, Glibc, Gzip, Make, Xz
테스트 스위트에 필요한 패키지:	없음
이 패키지를 필요로 하는 패키지:	없음
선택적 의존성 패키지:	없음

부록 D. 부트 및 sysconfig 스크립트 version-20191031

이 부록의 스크립트는 일반적으로 해당 스크립트가 위치한 디렉토리별로 나열되어 있다. 순서는 /etc/rc.d/init.d, /etc/sysconfig, /etc/sysconfig/network-devices, /etc/sysconfig/network-devices/services 순이다. 각 절 내에서, 파일들은 일반적으로 호출되는 순서대로 나열되어 있다.

D.1. /etc/rc.d/init.d/rc

rc 스크립트는 init에 의해 호출되어 부팅 프로세스를 시작하는 첫 번째 스크립트이다.

```
#!/bin/bash
#############################################################################
# Begin rc
#
# Description : Main Run Level Control Script
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#             : DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
#############################################################################

. /lib/lsb/init-functions

print_error_msg()
{
  log_failure_msg
  # $i is set when called
  MSG="FAILURE:₩n₩nYou should not be reading this error message.₩n₩n"
  MSG="${MSG}It means that an unforeseen error took place in₩n"
  MSG="${MSG}${i},₩n"
  MSG="${MSG}which exited with a return value of ${error_value}.₩n"

  MSG="${MSG}If you're able to track this error down to a bug in one of₩n"
  MSG="${MSG}the files provided by the ${DISTRO_MINI} book,₩n"
  MSG="${MSG}please be so kind to inform us at ${DISTRO_CONTACT}.₩n"
  log_failure_msg "${MSG}"

  log_info_msg "Press Enter to continue..."
  wait_for_user
}

check_script_status()
{
  # $i is set when called
  if [ ! -f ${i} ]; then
    log_warning_msg "${i} is not a valid symlink."
    SCRIPT_STAT="1"
  fi

  if [ ! -x ${i} ]; then
```

```
    log_warning_msg "${i} is not executable, skipping."
    SCRIPT_STAT="1"
  fi
}

run()
{
  if [ -z $interactive ]; then
    ${1} ${2}
    return $?
  fi

  while true; do
    read -p "Run ${1} ${2} (Yes/no/continue)? " -n 1 runit
    echo

    case ${runit} in
      c | C)
        interactive=""
        ${i} ${2}
        ret=${?}
        break;
        ;;

      n | N)
        return 0
        ;;

      y | Y)
        ${i} ${2}
        ret=${?}
        break
        ;;
    esac
  done

  return $ret
}

# Read any local settings/overrides
[ -r /etc/sysconfig/rc.site ] && source /etc/sysconfig/rc.site

DISTRO=${DISTRO:-"Linux From Scratch"}
DISTRO_CONTACT=${DISTRO_CONTACT:-"lfs-dev@linuxfromscratch.org (Registration required)"}
DISTRO_MINI=${DISTRO_MINI:-"LFS"}
IPROMPT=${IPROMPT:-"no"}

# These 3 signals will not cause our script to exit
trap "" INT QUIT TSTP

[ "${1}" != "" ] && runlevel=${1}

if [ "${runlevel}" == "" ]; then
  echo "Usage: ${0} <runlevel>" >&2
  exit 1
fi
```

```
previous=${PREVLEVEL}
[ "${previous}" == "" ] && previous=N

if [ ! -d /etc/rc.d/rc${runlevel}.d ]; then
  log_info_msg "/etc/rc.d/rc${runlevel}.d does not exist.₩n"
  exit 1
fi

if [ "$runlevel" == "6" -o "$runlevel" == "0" ]; then IPROMPT="no"; fi

# Note: In ${LOGLEVEL:-7}, it is ':' 'dash' '7', not minus 7
if [ "$runlevel" == "S" ]; then
  [ -r /etc/sysconfig/console ] && source /etc/sysconfig/console
  dmesg -n "${LOGLEVEL:-7}"
fi

if [ "${IPROMPT}" == "yes" -a "${runlevel}" == "S" ]; then
  # The total length of the distro welcome string, without escape codes
  wlen=${wlen:-$(echo "Welcome to ${DISTRO}" | wc -c )}
  welcome_message=${welcome_message:-"Welcome to ${INFO}${DISTRO}${NORMAL}"}

  # The total length of the interactive string, without escape codes
  ilen=${ilen:-$(echo "Press 'I' to enter interactive startup" | wc -c )}
  i_message=${i_message:-"Press '${FAILURE}I${NORMAL}' to enter interactive startup"}


  # dcol and icol are spaces before the message to center the message
  # on screen. itime is the amount of wait time for the user to press a key
  wcol=$(( ( ${COLUMNS} - ${wlen} ) / 2 ))
  icol=$(( ( ${COLUMNS} - ${ilen} ) / 2 ))
  itime=${itime:-"3"}

  echo -e "₩n₩n"
  echo -e "₩₩033[${wcol}G${welcome_message}"
  echo -e "₩₩033[${icol}G${i_message}${NORMAL}"
  echo ""
  read -t "${itime}" -n 1 interactive 2>&1 > /dev/null
fi

# Make lower case
[ "${interactive}" == "I" ] && interactive="i"
[ "${interactive}" != "i" ] && interactive=""

# Read the state file if it exists from runlevel S
[ -r /var/run/interactive ] && source /var/run/interactive

# Attempt to stop all services started by the previous runlevel,
# and killed in this runlevel
if [ "${previous}" != "N" ]; then
  for i in $(ls -v /etc/rc.d/rc${runlevel}.d/K* 2> /dev/null)
  do
    check_script_status
    if [ "${SCRIPT_STAT}" == "1" ]; then
      SCRIPT_STAT="0"
      continue
    fi
```

```
    suffix=${i#/etc/rc.d/rc$runlevel.d/K[0-9][0-9]}
    prev_start=/etc/rc.d/rc$previous.d/S[0-9][0-9]$suffix
    sysinit_start=/etc/rc.d/rcS.d/S[0-9][0-9]$suffix

    if [ "${runlevel}" != "0" -a "${runlevel}" != "6" ]; then
      if [ ! -f ${prev_start} -a  ! -f ${sysinit_start} ]; then
        MSG="WARNING:₩n₩n${i} can't be "
        MSG="${MSG}executed because it was not "
        MSG="${MSG}not started in the previous "
        MSG="${MSG}runlevel (${previous})."
        log_warning_msg "$MSG"
        continue
      fi
    fi

    run ${i} stop
    error_value=${?}

    if [ "${error_value}" != "0" ]; then print_error_msg; fi
  done
fi

if [ "${previous}" == "N" ]; then export IN_BOOT=1; fi

if [ "$runlevel" == "6" -a -n "${FASTBOOT}" ]; then
  touch /fastboot
fi


# Start all functions in this runlevel
for i in $( ls -v /etc/rc.d/rc${runlevel}.d/S* 2> /dev/null)
do
  if [ "${previous}" != "N" ]; then
    suffix=${i#/etc/rc.d/rc$runlevel.d/S[0-9][0-9]}
    stop=/etc/rc.d/rc$runlevel.d/K[0-9][0-9]$suffix
    prev_start=/etc/rc.d/rc$previous.d/S[0-9][0-9]$suffix

    [ -f ${prev_start} -a ! -f ${stop} ] && continue
  fi

  check_script_status
    if [ "${SCRIPT_STAT}" == "1" ]; then
      SCRIPT_STAT="0"
      continue
    fi

  case ${runlevel} in
    0|6)
      run ${i} stop
      ;;
    *)
      run ${i} start
      ;;
  esac

  error_value=${?}
```

```
  if [ "${error_value}" != "0" ]; then print_error_msg; fi
done

# Store interactive variable on switch from runlevel S and remove if not
if [ "${runlevel}" == "S" -a "${interactive}" == "i" ]; then
   echo "interactive=₩"i₩"" > /var/run/interactive
else
   rm -f /var/run/interactive 2> /dev/null
fi

# Copy the boot log on initial boot only
if [ "${previous}" == "N" -a  "${runlevel}" != "S" ]; then
  cat $BOOTLOG >> /var/log/boot.log

  # Mark the end of boot
  echo "--------" >> /var/log/boot.log

  # Remove the temporary file
  rm -f $BOOTLOG 2> /dev/null
fi

# End rc
```

D.2. /lib/lsb/init-functions

```
#!/bin/sh
########################################################################
#
# Begin /lib/lsb/init-funtions
#
# Description : Run Level Control Functions
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#             : DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : With code based on Matthias Benkmann's simpleinit-msb
#               http://winterdrache.de/linux/newboot/index.html
#
#               The file should be located in /lib/lsb
#
########################################################################

## Environmental setup
# Setup default values for environment
umask 022
export PATH="/bin:/usr/bin:/sbin:/usr/sbin"

## Set color commands, used via echo
# Please consult `man console_codes for more information
# under the "ECMA-48 Set Graphics Rendition" section
#
# Warning: when switching from a 8bit to a 9bit font,
# the linux console will reinterpret the bold (1;) to
```

```
# the top 256 glyphs of the 9bit font.  This does
# not affect framebuffer consoles

NORMAL="\\033[0;39m"        # Standard console grey
SUCCESS="\\033[1;32m"       # Success is green
WARNING="\\033[1;33m"       # Warnings are yellow
FAILURE="\\033[1;31m"       # Failures are red
INFO="\\033[1;36m"          # Information is light cyan
BRACKET="\\033[1;34m"       # Brackets are blue

# Use a colored prefix
BMPREFIX="     "
SUCCESS_PREFIX="${SUCCESS}  *  ${NORMAL} "
FAILURE_PREFIX="${FAILURE}*****${NORMAL} "
WARNING_PREFIX="${WARNING} *** ${NORMAL} "
SKIP_PREFIX="${INFO}  S   ${NORMAL}"

SUCCESS_SUFFIX="${BRACKET}[${SUCCESS}  OK  ${BRACKET}]${NORMAL}"
FAILURE_SUFFIX="${BRACKET}[${FAILURE} FAIL ${BRACKET}]${NORMAL}"
WARNING_SUFFIX="${BRACKET}[${WARNING} WARN ${BRACKET}]${NORMAL}"
SKIP_SUFFIX="${BRACKET}[${INFO} SKIP ${BRACKET}]${NORMAL}"

BOOTLOG=/run/bootlog
KILLDELAY=3
SCRIPT_STAT="0"

# Set any user specified environment variables e.g. HEADLESS
[ -r /etc/sysconfig/rc.site ]  && . /etc/sysconfig/rc.site

## Screen Dimensions
# Find current screen size
if [ -z "${COLUMNS}" ]; then
   COLUMNS=$(stty size)
   COLUMNS=${COLUMNS##* }
fi

# When using remote connections, such as a serial port, stty size returns 0
if [ "${COLUMNS}" = "0" ]; then
   COLUMNS=80
fi

## Measurements for positioning result messages
COL=$((${COLUMNS} - 8))
WCOL=$((${COL} - 2))

## Set Cursor Position Commands, used via echo
SET_COL="\\033[${COL}G"      # at the $COL char
SET_WCOL="\\033[${WCOL}G"    # at the $WCOL char
CURS_UP="\\033[1A\\033[0G"   # Up one line, at the 0'th char
CURS_ZERO="\\033[0G"

##############################################################################
# start_daemon()                                             #
# Usage: start_daemon [-f] [-n nicelevel] [-p pidfile] pathname [args...]     #
#                                                            #
# Purpose: This runs the specified program as a daemon                        #
#                                                            #
```

```
# Inputs: -f: (force) run the program even if it is already running.          #
#         -n nicelevel: specify a nice level. See 'man nice(1)'.              #
#         -p pidfile: use the specified file to determine PIDs.               #
#         pathname: the complete path to the specified program                #
#         args: additional arguments passed to the program (pathname)         #
#                                                                             #
# Return values (as defined by LSB exit codes):                              #
#       0 - program is running or service is OK                               #
#       1 - generic or unspecified error                                      #
#       2 - invalid or excessive argument(s)                                  #
#       5 - program is not installed                                          #
###############################################################################
start_daemon()
{
   local force=""
   local nice="0"
   local pidfile=""
   local pidlist=""
   local retval=""

   # Process arguments
   while true
   do
      case "${1}" in

         -f)
            force="1"
            shift 1
            ;;

         -n)
            nice="${2}"
            shift 2
            ;;

         -p)
            pidfile="${2}"
            shift 2
            ;;

         -*)
            return 2
            ;;

         *)
            program="${1}"
            break
            ;;
      esac
   done

   # Check for a valid program
   if [ ! -e "${program}" ]; then return 5; fi

   # Execute
   if [ -z "${force}" ]; then
      if [ -z "${pidfile}" ]; then
```

```
        # Determine the pid by discovery
        pidlist=`pidofproc "${1}"`
        retval="${?}"
      else
        # The PID file contains the needed PIDs
        # Note that by LSB requirement, the path must be given to pidofproc,
        # however, it is not used by the current implementation or standard.
        pidlist=`pidofproc -p "${pidfile}" "${1}"`
        retval="${?}"
      fi

      # Return a value ONLY
      # It is the init script's (or distribution's functions) responsibilty
      # to log messages!
      case "${retval}" in

        0)
          # Program is already running correctly, this is a
          # successful start.
          return 0
          ;;

        1)
          # Program is not running, but an invalid pid file exists
          # remove the pid file and continue
          rm -f "${pidfile}"
          ;;

        3)
          # Program is not running and no pidfile exists
          # do nothing here, let start_deamon continue.
          ;;

        *)
          # Others as returned by status values shall not be interpreted
          # and returned as an unspecified error.
          return 1
          ;;
      esac
    fi

    # Do the start!
    nice -n "${nice}" "${@}"
}

########################################################################
# killproc()                                                           #
# Usage: killproc [-p pidfile] pathname [signal]                       #
#                                                                      #
# Purpose: Send control signals to running processes                   #
#                                                                      #
# Inputs: -p pidfile, uses the specified pidfile                       #
#         pathname, pathname to the specified program                  #
#         signal, send this signal to pathname                         #
#                                                                      #
# Return values (as defined by LSB exit codes):                        #
#       0 - program (pathname) has stopped/is already stopped or a     #
```

```
#          running program has been sent specified signal and stopped        #
#          successfully                                                       #
#       1 - generic or unspecified error                                      #
#       2 - invalid or excessive argument(s)                                  #
#       5 - program is not installed                                          #
#       7 - program is not running and a signal was supplied                  #
#############################################################################
killproc()
{
   local pidfile
   local program
   local prefix
   local progname
   local signal="-TERM"
   local fallback="-KILL"
   local nosig
   local pidlist
   local retval
   local pid
   local delay="30"
   local piddead
   local dtime

   # Process arguments
   while true; do
      case "${1}" in
         -p)
            pidfile="${2}"
            shift 2
            ;;

          *)
             program="${1}"
             if [ -n "${2}" ]; then
                signal="${2}"
                fallback=""
             else
                nosig=1
             fi

             # Error on additional arguments
             if [ -n "${3}" ]; then
                return 2
             else
                break
             fi
             ;;
      esac
   done

   # Check for a valid program
   if [ ! -e "${program}" ]; then return 5; fi

   # Check for a valid signal
   check_signal "${signal}"
   if [ "${?}" -ne "0" ]; then return 2; fi
```

```
    # Get a list of pids
    if [ -z "${pidfile}" ]; then
       # determine the pid by discovery
       pidlist=`pidofproc "${1}"`
       retval="${?}"
    else
       # The PID file contains the needed PIDs
       # Note that by LSB requirement, the path must be given to pidofproc,
       # however, it is not used by the current implementation or standard.
       pidlist=`pidofproc -p "${pidfile}" "${1}"`
       retval="${?}"
    fi

    # Return a value ONLY
    # It is the init script's (or distribution's functions) responsibilty
    # to log messages!
    case "${retval}" in

       0)
          # Program is running correctly
          # Do nothing here, let killproc continue.
          ;;

       1)
          # Program is not running, but an invalid pid file exists
          # Remove the pid file.
          rm -f "${pidfile}"

          # This is only a success if no signal was passed.
          if [ -n "${nosig}" ]; then
             return 0
          else
             return 7
          fi
          ;;

       3)
          # Program is not running and no pidfile exists
          # This is only a success if no signal was passed.
          if [ -n "${nosig}" ]; then
             return 0
          else
             return 7
          fi
          ;;

       *)
          # Others as returned by status values shall not be interpreted
          # and returned as an unspecified error.
          return 1
          ;;
    esac

    # Perform different actions for exit signals and control signals
    check_sig_type "${signal}"

    if [ "${?}" -eq "0" ]; then # Signal is used to terminate the program
```

```
    # Account for empty pidlist (pid file still exists and no
    # signal was given)
    if [ "${pidlist}" != "" ]; then

       # Kill the list of pids
       for pid in ${pidlist}; do

          kill -0 "${pid}" 2> /dev/null

          if [ "${?}" -ne "0" ]; then
             # Process is dead, continue to next and assume all is well
             continue
          else
             kill "${signal}" "${pid}" 2> /dev/null

             # Wait up to ${delay}/10 seconds to for "${pid}" to
             # terminate in 10ths of a second

             while [ "${delay}" -ne "0" ]; do
                kill -0 "${pid}" 2> /dev/null || piddead="1"
                if [ "${piddead}" = "1" ]; then break; fi
                sleep 0.1
                delay="$(( ${delay} - 1 ))"
             done

             # If a fallback is set, and program is still running, then
             # use the fallback
             if [ -n "${fallback}" -a "${piddead}" != "1" ]; then
                kill "${fallback}" "${pid}" 2> /dev/null
                sleep 1
                # Check again, and fail if still running
                kill -0 "${pid}" 2> /dev/null && return 1
             fi
          fi
       done
    fi

    # Check for and remove stale PID files.
    if [ -z "${pidfile}" ]; then
       # Find the basename of $program
       prefix=`echo "${program}" | sed 's/[^/]*$//'`
       progname=`echo "${program}" | sed "s@${prefix}@@"`

       if [ -e "/var/run/${progname}.pid" ]; then
          rm -f "/var/run/${progname}.pid" 2> /dev/null
       fi
    else
       if [ -e "${pidfile}" ]; then rm -f "${pidfile}" 2> /dev/null; fi
    fi

# For signals that do not expect a program to exit, simply
# let kill do its job, and evaluate kill's return for value

else # check_sig_type - signal is not used to terminate program
    for pid in ${pidlist}; do
       kill "${signal}" "${pid}"
```

```
          if [ "${?}" -ne "0" ]; then return 1; fi
       done
    fi
}

########################################################################
# pidofproc()                                                          #
# Usage: pidofproc [-p pidfile] pathname                               #
#                                                                      #
# Purpose: This function returns one or more pid(s) for a particular daemon    #
#                                                                      #
# Inputs: -p pidfile, use the specified pidfile instead of pidof               #
#         pathname, path to the specified program                      #
#                                                                      #
# Return values (as defined by LSB status codes):                      #
#       0 - Success (PIDs to stdout)                                   #
#       1 - Program is dead, PID file still exists (remaining PIDs output)     #
#       3 - Program is not running (no output)                         #
########################################################################
pidofproc()
{
    local pidfile
    local program
    local prefix
    local progname
    local pidlist
    local lpids
    local exitstatus="0"

    # Process arguments
    while true; do
       case "${1}" in

          -p)
             pidfile="${2}"
             shift 2
             ;;

          *)
             program="${1}"
             if [ -n "${2}" ]; then
                # Too many arguments
                # Since this is status, return unknown
                return 4
             else
                break
             fi
             ;;
       esac
    done

    # If a PID file is not specified, try and find one.
    if [ -z "${pidfile}" ]; then
       # Get the program's basename
       prefix=`echo "${program}" | sed 's/[^/]*$//'`

       if [ -z "${prefix}" ]; then
```

```
        progname="${program}"
      else
        progname=`echo "${program}" | sed "s@${prefix}@@"`
      fi

      # If a PID file exists with that name, assume that is it.
      if [ -e "/var/run/${progname}.pid" ]; then
         pidfile="/var/run/${progname}.pid"
      fi
   fi

   # If a PID file is set and exists, use it.
   if [ -n "${pidfile}" -a -e "${pidfile}" ]; then

      # Use the value in the first line of the pidfile
      pidlist=`/bin/head -n1 "${pidfile}"`
      # This can optionally be written as 'sed 1q' to repalce 'head -n1'
      # should LFS move /bin/head to /usr/bin/head
   else
      # Use pidof
      pidlist=`pidof "${program}"`
   fi

   # Figure out if all listed PIDs are running.
   for pid in ${pidlist}; do
      kill -0 ${pid} 2> /dev/null

      if [ "${?}" -eq "0" ]; then
         lpids="${lpids}${pid} "
      else
         exitstatus="1"
      fi
   done

   if [ -z "${lpids}" -a ! -f "${pidfile}" ]; then
      return 3
   else
      echo "${lpids}"
      return "${exitstatus}"
   fi
}

########################################################################
# statusproc()                                                         #
# Usage: statusproc [-p pidfile] pathname                              #
#                                                                      #
# Purpose: This function prints the status of a particular daemon to stdout    #
#                                                                      #
# Inputs: -p pidfile, use the specified pidfile instead of pidof             #
#         pathname, path to the specified program                      #
#                                                                      #
# Return values:                                                       #
#       0 - Status printed                                             #
#       1 - Input error. The daemon to check was not specified.              #
########################################################################
statusproc()
{
```

```
  local pidfile
  local pidlist

  if [ "${#}" = "0" ]; then
    echo "Usage: statusproc [-p pidfle] {program}"
    exit 1
  fi

  # Process arguments
  while true; do
     case "${1}" in

        -p)
           pidfile="${2}"
           shift 2
           ;;

        *)
           if [ -n "${2}" ]; then
              echo "Too many arguments"
              return 1
           else
              break
           fi
           ;;
     esac
  done

  if [ -n "${pidfile}" ]; then
    pidlist=`pidofproc -p "${pidfile}" $@`
  else
    pidlist=`pidofproc $@`
  fi

  # Trim trailing blanks
  pidlist=`echo "${pidlist}" | sed -r 's/ +$//'`

  base="${1##*/}"

  if [ -n "${pidlist}" ]; then
    /bin/echo -e "${INFO}${base} is running with Process" ₩
      "ID(s) ${pidlist}.${NORMAL}"
  else
    if [ -n "${base}" -a -e "/var/run/${base}.pid" ]; then
      /bin/echo -e "${WARNING}${1} is not running but" ₩
        "/var/run/${base}.pid exists.${NORMAL}"
    else
      if [ -n "${pidfile}" -a -e "${pidfile}" ]; then
        /bin/echo -e "${WARNING}${1} is not running" ₩
          "but ${pidfile} exists.${NORMAL}"
      else
        /bin/echo -e "${INFO}${1} is not running.${NORMAL}"
      fi
    fi
  fi
}
```

```
########################################################################
# timespec()                                                           #
#                                                                      #
# Purpose: An internal utility function to format a timestamp          #
#          a boot log file.  Sets the STAMP variable.                  #
#                                                                      #
# Return value: Not used                                               #
########################################################################
timespec()
{
  STAMP="$(echo `date +"%b %d %T %:z"` `hostname`) "
  return 0
}

########################################################################
# log_success_msg()                                                    #
# Usage: log_success_msg ["message"]                                   #
#                                                                      #
# Purpose: Print a successful status message to the screen and         #
#          a boot log file.                                            #
#                                                                      #
# Inputs: $@ - Message                                                 #
#                                                                      #
# Return values: Not used                                              #
########################################################################
log_success_msg()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

   # Strip non-printable characters from log file
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`

   timespec
   /bin/echo -e "${STAMP} ${logmessage} OK" >> ${BOOTLOG}

   return 0
}

log_success_msg2()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${SUCCESS_PREFIX}${SET_COL}${SUCCESS_SUFFIX}"

   echo " OK" >> ${BOOTLOG}

   return 0
}

########################################################################
# log_failure_msg()                                                    #
# Usage: log_failure_msg ["message"]                                   #
#                                                                      #
# Purpose: Print a failure status message to the screen and            #
#          a boot log file.                                            #
#                                                                      #
# Inputs: $@ - Message                                                 #
```

```
#                                                     #
# Return values: Not used                                   #
#########################################################################
log_failure_msg()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

   # Strip non-printable characters from log file

   timespec
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`
   /bin/echo -e "${STAMP} ${logmessage} FAIL" >> ${BOOTLOG}

   return 0
}

log_failure_msg2()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${FAILURE_PREFIX}${SET_COL}${FAILURE_SUFFIX}"

   echo "FAIL" >> ${BOOTLOG}

   return 0
}

#########################################################################
# log_warning_msg()                                   #
# Usage: log_warning_msg ["message"]                        #
#                                                     #
# Purpose: Print a warning status message to the screen and       #
#         a boot log file.                            #
#                                                     #
# Return values: Not used                                   #
#########################################################################
log_warning_msg()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${WARNING_PREFIX}${SET_COL}${WARNING_SUFFIX}"

   # Strip non-printable characters from log file
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`
   timespec
   /bin/echo -e "${STAMP} ${logmessage} WARN" >> ${BOOTLOG}

   return 0
}

log_skip_msg()
{
   /bin/echo -n -e "${BMPREFIX}${@}"
   /bin/echo -e "${CURS_ZERO}${SKIP_PREFIX}${SET_COL}${SKIP_SUFFIX}"

   # Strip non-printable characters from log file
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`
   /bin/echo "SKIP" >> ${BOOTLOG}
```

```
   return 0
}

########################################################################
# log_info_msg()                                                       #
# Usage: log_info_msg message                                          #
#                                                                      #
# Purpose: Print an information message to the screen and              #
#         a boot log file.  Does not print a trailing newline character.      #
#                                                                      #
# Return values: Not used                                              #
########################################################################
log_info_msg()
{
   /bin/echo -n -e "${BMPREFIX}${@}"

   # Strip non-printable characters from log file
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`
   timespec
   /bin/echo -n -e "${STAMP} ${logmessage}" >> ${BOOTLOG}

   return 0
}

log_info_msg2()
{
   /bin/echo -n -e "${@}"

   # Strip non-printable characters from log file
   logmessage=`echo "${@}" | sed 's/₩₩₩033[^a-zA-Z]*.//g'`
   /bin/echo -n -e "${logmessage}" >> ${BOOTLOG}

   return 0
}

########################################################################
# evaluate_retval()                                                    #
# Usage: Evaluate a return value and print success or failyure as appropriate  #
#                                                                      #
# Purpose: Convenience function to terminate an info message                 #
#                                                                      #
# Return values: Not used                                              #
########################################################################
evaluate_retval()
{
  local error_value="${?}"

  if [ ${error_value} = 0 ]; then
    log_success_msg2
  else
    log_failure_msg2
  fi
}

########################################################################
# check_signal()                                                       #
```

```
# Usage: check_signal [ -{signal} | {signal} ]                          #
#                                                                       #
# Purpose: Check for a valid signal.  This is not defined by any LSB draft,    #
#          however, it is required to check the signals to determine if the    #
#          signals chosen are invalid arguments to the other functions.        #
#                                                                       #
# Inputs: Accepts a single string value in the form or -{signal} or {signal}   #
#                                                                       #
# Return values:                                                        #
#       0 - Success (signal is valid                                    #
#       1 - Signal is not valid                                         #
#########################################################################
check_signal()
{
   local valsig

   # Add error handling for invalid signals
   valsig="-ALRM -HUP -INT -KILL -PIPE -POLL -PROF -TERM -USR1 -USR2"
   valsig="${valsig} -VTALRM -STKFLT -PWR -WINCH -CHLD -URG -TSTP -TTIN"
   valsig="${valsig} -TTOU -STOP -CONT -ABRT -FPE -ILL -QUIT -SEGV -TRAP"
   valsig="${valsig} -SYS -EMT -BUS -XCPU -XFSZ -0 -1 -2 -3 -4 -5 -6 -8 -9"
   valsig="${valsig} -11 -13 -14 -15"

   echo "${valsig}" | grep -- " ${1} " > /dev/null

   if [ "${?}" -eq "0" ]; then
      return 0
   else
      return 1
   fi
}

#########################################################################
# check_sig_type()                                                      #
# Usage: check_signal [ -{signal} | {signal} ]                          #
#                                                                       #
# Purpose: Check if signal is a program termination signal or a control signal #
#          This is not defined by any LSB draft, however, it is required to    #
#          check the signals to determine if they are intended to end a        #
#          program or simply to control it.                                    #
#                                                                       #
# Inputs: Accepts a single string value in the form or -{signal} or {signal}   #
#                                                                       #
# Return values:                                                        #
#       0 - Signal is used for program termination                      #
#       1 - Signal is used for program control                          #
#########################################################################
check_sig_type()
{
   local valsig

   # The list of termination signals (limited to generally used items)
   valsig="-ALRM -INT -KILL -TERM -PWR -STOP -ABRT -QUIT -2 -3 -6 -9 -14 -15"

   echo "${valsig}" | grep -- " ${1} " > /dev/null

   if [ "${?}" -eq "0" ]; then
```

```
      return 0
   else
      return 1
   fi
}

########################################################################
# wait_for_user()                                                      #
#                                                                      #
# Purpose: Wait for the user to respond if not a headless system       #
#                                                                      #
########################################################################
wait_for_user()
{
  # Wait for the user by default
  [ "${HEADLESS=0}" = "0" ] && read ENTER
  return 0
}

########################################################################
# is_true()                                                            #
#                                                                      #
# Purpose: Utility to test if a variable is true | yes | 1            #
#                                                                      #
########################################################################
is_true()
{
  [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ] ||  [ "$1" = "y" ] ||
  [ "$1" = "t" ]
}

# End /lib/lsb/init-functions
```

D.3. /etc/rc.d/init.d/mountvirtfs

```
#!/bin/sh
########################################################################
# Begin mountvirtfs
#
# Description : Mount proc, sysfs, and run
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           mountvirtfs
# Required-Start:     $first
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:      S
```

```
# Default-Stop:
# Short-Description:   Mounts /sys and /proc virtual (kernel) filesystems.
#                      Mounts /run (tmpfs) and /dev (devtmpfs).
# Description:         Mounts /sys and /proc virtual (kernel) filesystems.
#                      Mounts /run (tmpfs) and /dev (devtmpfs).
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    # Make sure /run is available before logging any messages
    if ! mountpoint /run >/dev/null; then
      mount /run || failed=1
    fi

    mkdir -p /run/lock /run/shm
    chmod 1777 /run/shm /run/lock

    log_info_msg "Mounting virtual file systems: ${INFO}/run"

    if ! mountpoint /proc >/dev/null; then
      log_info_msg2 " ${INFO}/proc"
      mount -o nosuid,noexec,nodev /proc || failed=1
    fi

    if ! mountpoint /sys >/dev/null; then
      log_info_msg2 " ${INFO}/sys"
      mount -o nosuid,noexec,nodev /sys || failed=1
    fi

    if ! mountpoint /dev >/dev/null; then
      log_info_msg2 " ${INFO}/dev"
      mount -o mode=0755,nosuid /dev  || failed=1
    fi

    ln -sfn /run/shm /dev/shm

    (exit ${failed})
    evaluate_retval
    exit $failed
    ;;

  *)
    echo "Usage: ${0} {start}"
    exit 1
    ;;
esac

# End mountvirtfs
```

D.4. /etc/rc.d/init.d/modules

```
#!/bin/sh
########################################################################
```

```
# Begin modules
#
# Description : Module auto-loading script
#
# Authors     : Zack Winkles
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           modules
# Required-Start:     mountvirtfs
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:
# Short-Description:  Loads required modules.
# Description:        Loads modules listed in /etc/sysconfig/modules.
# X-LFS-Provided-By:  LFS
### END INIT INFO

# Assure that the kernel has module support.
[ -e /proc/modules ] || exit 0

. /lib/lsb/init-functions

case "${1}" in
  start)
    # Exit if there's no modules file or there are no
    # valid entries
    [ -r /etc/sysconfig/modules ]             || exit 0
    egrep -qv '^($|#)' /etc/sysconfig/modules || exit 0

    log_info_msg "Loading modules:"

    # Only try to load modules if the user has actually given us
    # some modules to load.

    while read module args; do

      # Ignore comments and blank lines.
      case "$module" in
        ""|"#"*) continue ;;
      esac

      # Attempt to load the module, passing any arguments provided.
      modprobe ${module} ${args} >/dev/null

      # Print the module name if successful, otherwise take note.
      if [ $? -eq 0 ]; then
        log_info_msg2 " ${module}"
      else
        failedmod="${failedmod} ${module}"
```

```
      fi
    done < /etc/sysconfig/modules

    # Print a message about successfully loaded modules on the correct line.
    log_success_msg2

    # Print a failure message with a list of any modules that
    # may have failed to load.
    if [ -n "${failedmod}" ]; then
      log_failure_msg "Failed to load modules:${failedmod}"
      exit 1
    fi
    ;;

  *)
    echo "Usage: ${0} {start}"
    exit 1
    ;;
esac

exit 0

# End modules
```

D.5. /etc/rc.d/init.d/udev

```
#!/bin/sh
########################################################################
# Begin udev
#
# Description : Udev cold-plugging script
#
# Authors     : Zack Winkles, Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           udev $time
# Required-Start:     localnet
# Should-Start:       modules
# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:
# Short-Description:  Populates /dev with device nodes.
# Description:        Mounts a tempfs on /dev and starts the udevd daemon.
#                     Device nodes are created as defined by udev.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions
```

```
case "${1}" in
  start)
    log_info_msg "Populating /dev with device nodes... "
    if ! grep -q '[[:space:]]sysfs' /proc/mounts; then
      log_failure_msg2
      msg="FAILURE:₩n₩nUnable to create "
      msg="${msg}devices without a SysFS filesystem₩n₩n"
      msg="${msg}After you press Enter, this system "
      msg="${msg}will be halted and powered off.₩n₩n"
      log_info_msg "$msg"
      log_info_msg "Press Enter to continue..."
      wait_for_user
      /etc/rc.d/init.d/halt stop
    fi

    # Start the udev daemon to continually watch for, and act on,
    # uevents
    /sbin/udevd --daemon

    # Now traverse /sys in order to "coldplug" devices that have
    # already been discovered
    /sbin/udevadm trigger --action=add    --type=subsystems
    /sbin/udevadm trigger --action=add    --type=devices
    /sbin/udevadm trigger --action=change --type=devices

    # Now wait for udevd to process the uevents we triggered
    if ! is_true "$OMIT_UDEV_SETTLE"; then
      /sbin/udevadm settle
    fi

    # If any LVM based partitions are on the system, ensure they
    # are activated so they can be used.
    if [ -x /sbin/vgchange ]; then /sbin/vgchange -a y >/dev/null; fi

    log_success_msg2
    ;;

  *)
    echo "Usage ${0} {start}"
    exit 1
    ;;
esac

exit 0

# End udev
```

D.6. /etc/rc.d/init.d/swap

```
#!/bin/sh
########################################################################
# Begin swap
#
# Description : Swap Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
```

```
#             DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           swap
# Required-Start:     udev
# Should-Start:       modules
# Required-Stop:      localnet $local_fs
# Should-Stop:
# Default-Start:      S
# Default-Stop:       0 6
# Short-Description:  Mounts and unmounts swap partitions.
# Description:        Mounts and unmounts swap partitions defined in
#                     /etc/fstab.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "Activating all swap files/partitions..."
    swapon -a
    evaluate_retval
    ;;

  stop)
    log_info_msg "Deactivating all swap files/partitions..."
    swapoff -a
    evaluate_retval
    ;;

  restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;

  status)
    log_success_msg "Retrieving swap status."
    swapon -s
    ;;

  *)
    echo "Usage: ${0} {start|stop|restart|status}"
    exit 1
    ;;
esac

exit 0

# End swap
```

D.7. /etc/rc.d/init.d/setclock

```
#!/bin/sh
########################################################################
# Begin setclock
#
# Description : Setting Linux Clock
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:
# Required-Start:
# Should-Start:       modules
# Required-Stop:
# Should-Stop:        $syslog
# Default-Start:      S
# Default-Stop:
# Short-Description:   Stores and restores time from the hardware clock
# Description:        On boot, system time is obtained from hwclock.  The
#                     hardware clock can also be set on shutdown.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

[ -r /etc/sysconfig/clock ] && . /etc/sysconfig/clock


case "${UTC}" in
  yes|true|1)
    CLOCKPARAMS="${CLOCKPARAMS} --utc"
    ;;

  no|false|0)
    CLOCKPARAMS="${CLOCKPARAMS} --localtime"
    ;;

esac

case ${1} in
  start)
    hwclock --hctosys ${CLOCKPARAMS} >/dev/null
    ;;

  stop)
    log_info_msg "Setting hardware clock..."
    hwclock --systohc ${CLOCKPARAMS} >/dev/null
    evaluate_retval
    ;;
```

```
  *)
    echo "Usage: ${0} {start|stop}"
    exit 1
    ;;

esac

exit 0
```

D.8. /etc/rc.d/init.d/checkfs

```
#!/bin/sh
########################################################################
# Begin checkfs
#
# Description : File System Check
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#             A. Luebke - luebke@users.sourceforge.net
#             DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Based on checkfs script from LFS-3.1 and earlier.
#
# From man fsck
# 0    - No errors
# 1    - File system errors corrected
# 2    - System should be rebooted
# 4    - File system errors left uncorrected
# 8    - Operational error
# 16   - Usage or syntax error
# 32   - Fsck canceled by user request
# 128  - Shared library error
#
########################################################################

### BEGIN INIT INFO
# Provides:           checkfs
# Required-Start:     udev
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:
# Short-Description:  Checks local filesystems before mounting.
# Description:        Checks local filesystmes before mounting.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
```

```
if [ -f /fastboot ]; then
   msg="/fastboot found, will omit "
   msg="${msg} file system checks as requested.₩n"
   log_info_msg "${msg}"
   exit 0
fi

log_info_msg "Mounting root file system in read-only mode... "
mount -n -o remount,ro / >/dev/null

if [ ${?} != 0 ]; then
   log_failure_msg2
   msg="₩n₩nCannot check root "
   msg="${msg}filesystem because it could not be mounted "
   msg="${msg}in read-only mode.₩n₩n"
   msg="${msg}After you press Enter, this system will be "
   msg="${msg}halted and powered off.₩n₩n"
   log_failure_msg "${msg}"

   log_info_msg "Press Enter to continue..."
   wait_for_user
   /etc/rc.d/init.d/halt stop
else
   log_success_msg2
fi

if [ -f /forcefsck ]; then
   msg="/forcefsck found, forcing file"
   msg="${msg} system checks as requested."
   log_success_msg "$msg"
   options="-f"
else
   options=""
fi

log_info_msg "Checking file systems..."
# Note: -a option used to be -p; but this fails e.g. on fsck.minix
if is_true "$VERBOSE_FSCK"; then
  fsck ${options} -a -A -C -T
else
  fsck ${options} -a -A -C -T >/dev/null
fi

error_value=${?}

if [ "${error_value}" = 0 ]; then
   log_success_msg2
fi

if [ "${error_value}" = 1 ]; then
   msg="₩nWARNING:₩n₩nFile system errors "
   msg="${msg}were found and have been corrected.₩n"
   msg="${msg}     You may want to double-check that "
   msg="${msg}everything was fixed properly."
   log_warning_msg "$msg"
fi
```

```
    if [ "${error_value}" = 2 -o "${error_value}" = 3 ]; then
      msg="₩nWARNING:₩n₩nFile system errors "
      msg="${msg}were found and have been been "
      msg="${msg}corrected, but the nature of the "
      msg="${msg}errors require this system to be rebooted.₩n₩n"
      msg="${msg}After you press enter, "
      msg="${msg}this system will be rebooted₩n₩n"
      log_failure_msg "$msg"

      log_info_msg "Press Enter to continue..."
      wait_for_user
      reboot -f
    fi

    if [ "${error_value}" -gt 3 -a "${error_value}" -lt 16 ]; then
      msg="₩nFAILURE:₩n₩nFile system errors "
      msg="${msg}were encountered that could not be "
      msg="${msg}fixed automatically.₩nThis system "
      msg="${msg}cannot continue to boot and will "
      msg="${msg}therefore be halted until those "
      msg="${msg}errors are fixed manually by a "
      msg="${msg}System Administrator.₩n₩n"
      msg="${msg}After you press Enter, this system will be "
      msg="${msg}halted and powered off.₩n₩n"
      log_failure_msg "$msg"

      log_info_msg "Press Enter to continue..."
      wait_for_user
      /etc/rc.d/init.d/halt stop
    fi

    if [ "${error_value}" -ge 16 ]; then
      msg="FAILURE:₩n₩nUnexpected failure "
      msg="${msg}running fsck.  Exited with error "
      msg="${msg} code: ${error_value}.₩n"
      log_info_msg $msg
      exit ${error_value}
    fi

    exit 0
    ;;
  *)
    echo "Usage: ${0} {start}"
    exit 1
    ;;
esac

# End checkfs
```

D.9. /etc/rc.d/init.d/mountfs

```
#!/bin/sh
########################################################################
# Begin mountfs
#
# Description : File System Mount Script
```

```
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
###############################################################################

### BEGIN INIT INFO
# Provides:           $local_fs
# Required-Start:     udev checkfs
# Should-Start:       modules
# Required-Stop:      localnet
# Should-Stop:
# Default-Start:      S
# Default-Stop:       0 6
# Short-Description:  Mounts/unmounts local filesystems defined in /etc/fstab.
# Description:        Remounts root filesystem read/write and mounts all
#                     remaining local filesystems defined in /etc/fstab on
#                     start.  Remounts root filesystem read-only and unmounts
#                     remaining filesystems on stop.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "Remounting root file system in read-write mode..."
    mount --options remount,rw / >/dev/null
    evaluate_retval

    # Remove fsck-related file system watermarks.
    rm -f /fastboot /forcefsck

    # Make sure /dev/pts exists
    mkdir -p /dev/pts

    # This will mount all filesystems that do not have _netdev in
    # their option list.  _netdev denotes a network filesystem.

    log_info_msg "Mounting remaining file systems..."
    mount --all --test-opts no_netdev >/dev/null
    evaluate_retval
    exit $failed
    ;;

  stop)
    # Don't unmount virtual file systems like /run
    log_info_msg "Unmounting all other currently mounted file systems..."
    # Ensure any loop devies are removed
    losetup -D
    umount --all --detach-loop --read-only ₩
          --types notmpfs,nosysfs,nodevtmpfs,noproc,nodevpts >/dev/null
    evaluate_retval
```

```
    # Make sure / is mounted read only (umount bug)
    mount --options remount,ro /

    # Make all LVM volume groups unavailable, if appropriate
    # This fails if swap or / are on an LVM partition
    #if [ -x /sbin/vgchange ]; then /sbin/vgchange -an > /dev/null; fi
    ;;

  *)
    echo "Usage: ${0} {start|stop}"
    exit 1
    ;;
esac

# End mountfs
```

D.10. /etc/rc.d/init.d/udev_retry

```
#!/bin/sh
########################################################################
# Begin udev_retry
#
# Description : Udev cold-plugging script (retry)
#
# Authors     : Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#               Bryan Kadzban -
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           udev_retry
# Required-Start:     udev
# Should-Start:       $local_fs cleanfs
# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:
# Short-Description:  Replays failed uevents and creates additional devices.
# Description:        Replays any failed uevents that were skipped due to
#                     slow hardware initialization, and creates those needed
#                     device nodes
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "Retrying failed uevents, if any..."

    # As of udev-186, the --run option is no longer valid
    #rundir=$(/sbin/udevadm info --run)
```

```
    rundir=/run/udev
    # From Debian: "copy the rules generated before / was mounted
    # read-write":

    for file in ${rundir}/tmp-rules--*; do
      dest=${file##*tmp-rules--}
      [ "$dest" = '*' ] && break
      cat $file >> /etc/udev/rules.d/$dest
      rm -f $file
    done

    # Re-trigger the uevents that may have failed,
    # in hope they will succeed now
    /bin/sed -e 's/#.*$//' /etc/sysconfig/udev_retry | /bin/grep -v '^$' | ₩
    while read line ; do
      for subsystem in $line ; do
        /sbin/udevadm trigger --subsystem-match=$subsystem --action=add
      done
    done

    # Now wait for udevd to process the uevents we triggered
    if ! is_true "$OMIT_UDEV_RETRY_SETTLE"; then
      /sbin/udevadm settle
    fi

    evaluate_retval
    ;;

  *)
    echo "Usage ${0} {start}"
    exit 1
    ;;
esac

exit 0

# End udev_retry
```

D.11. /etc/rc.d/init.d/cleanfs

```
#!/bin/sh
########################################################################
# Begin cleanfs
#
# Description : Clean file system
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           cleanfs
```

```
# Required-Start:      $local_fs
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Cleans temporary directories early in the boot process.
# Description:         Cleans temporary directories /var/run, /var/lock, and
#                      optionally, /tmp.  cleanfs also creates /var/run/utmp
#                      and any files defined in /etc/sysconfig/createfiles.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

# Function to create files/directory on boot.
create_files()
{
  # Input to file descriptor 9 and output to stdin (redirection)
  exec 9>&0 < /etc/sysconfig/createfiles

  while read name type perm usr grp dtype maj min junk
  do
    # Ignore comments and blank lines.
    case "${name}" in
      ""|₩#*) continue ;;
    esac

    # Ignore existing files.
    if [ ! -e "${name}" ]; then
      # Create stuff based on its type.
      case "${type}" in
        dir)
          mkdir "${name}"
          ;;
        file)
          :> "${name}"
          ;;
        dev)
          case "${dtype}" in
            char)
              mknod "${name}" c ${maj} ${min}
              ;;
            block)
              mknod "${name}" b ${maj} ${min}
              ;;
            pipe)
              mknod "${name}" p
              ;;
            *)
              log_warning_msg "₩nUnknown device type: ${dtype}"
              ;;
          esac
          ;;
        *)
          log_warning_msg "₩nUnknown type: ${type}"
          continue
```

```
            ;;
        esac

        # Set up the permissions, too.
        chown ${usr}:${grp} "${name}"
        chmod ${perm} "${name}"
      fi
    done

    # Close file descriptor 9 (end redirection)
    exec 0>&9 9>&-
    return 0
}

case "${1}" in
  start)
    log_info_msg "Cleaning file systems:"

    if [ "${SKIPTMPCLEAN}" = "" ]; then
      log_info_msg2 " /tmp"
      cd /tmp &&
      find . -xdev -mindepth 1 ! -name lost+found -delete || failed=1
    fi

    > /var/run/utmp

    if grep -q '^utmp:' /etc/group ; then
      chmod 664 /var/run/utmp
      chgrp utmp /var/run/utmp
    fi

    (exit ${failed})
    evaluate_retval

    if egrep -qv '^(#|$)' /etc/sysconfig/createfiles 2>/dev/null; then
      log_info_msg "Creating files and directories... "
      create_files      # Always returns 0
      evaluate_retval
    fi

    exit $failed
    ;;
  *)
    echo "Usage: ${0} {start}"
    exit 1
    ;;
esac

# End cleanfs
```

D.12. /etc/rc.d/init.d/console

```
#!/bin/sh
########################################################################
# Begin console
#
```

```
# Description : Sets keymap and screen font
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Alexander E. Patrakov
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
##############################################################################

### BEGIN INIT INFO
# Provides:           console
# Required-Start:     $local_fs
# Should-Start:       udev_retry
# Required-Stop:
# Should-Stop:
# Default-Start:      S
# Default-Stop:
# Short-Description:  Sets up a localised console.
# Description:        Sets up fonts and language settings for the user's
#                     local as defined by /etc/sysconfig/console.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

# Native English speakers probably don't have /etc/sysconfig/console at all
[ -r /etc/sysconfig/console ] && . /etc/sysconfig/console

is_true()
{
  [ "$1" = "1" ] || [ "$1" = "yes" ] || [ "$1" = "true" ]
}

failed=0

case "${1}" in
  start)
    # See if we need to do anything
    if [ -z "${KEYMAP}"         ] && [ -z "${KEYMAP_CORRECTIONS}" ] &&
       [ -z "${FONT}"           ] && [ -z "${LEGACY_CHARSET}"     ] &&
       ! is_true "${UNICODE}"; then
       exit 0
    fi

    # There should be no bogus failures below this line!
    log_info_msg "Setting up Linux console..."

    # Figure out if a framebuffer console is used
    [ -d /sys/class/graphics/fb0 ] && use_fb=1 || use_fb=0

    # Figure out the command to set the console into the
    # desired mode
    is_true "${UNICODE}" &&
      MODE_COMMAND="echo -en '₩033%G' && kbd_mode -u" ||
      MODE_COMMAND="echo -en '₩033%@₩033(K' && kbd_mode -a"
```

```
    # On framebuffer consoles, font has to be set for each vt in
    # UTF-8 mode. This doesn't hurt in non-UTF-8 mode also.

    ! is_true "${use_fb}" || [ -z "${FONT}" ] ||
      MODE_COMMAND="${MODE_COMMAND} && setfont ${FONT}"

    # Apply that command to all consoles mentioned in
    # /etc/inittab. Important: in the UTF-8 mode this should
    # happen before setfont, otherwise a kernel bug will
    # show up and the unicode map of the font will not be
    # used.

    for TTY in `grep '^[^#].*respawn:/sbin/agetty' /etc/inittab |
      grep -o '₩btty[[:digit:]]*₩b'`
    do
      openvt -f -w -c ${TTY#tty} -- ₩
        /bin/sh -c "${MODE_COMMAND}" || failed=1
    done

    # Set the font (if not already set above) and the keymap
    [ "${use_fb}" == "1" ] || [ -z "${FONT}" ] || setfont $FONT || failed=1

    [ -z "${KEYMAP}" ] ||
      loadkeys ${KEYMAP} >/dev/null 2>&1 ||
      failed=1

    [ -z "${KEYMAP_CORRECTIONS}" ] ||
      loadkeys ${KEYMAP_CORRECTIONS} >/dev/null 2>&1 ||
      failed=1

    # Convert the keymap from $LEGACY_CHARSET to UTF-8
    [ -z "$LEGACY_CHARSET" ] ||
      dumpkeys -c "$LEGACY_CHARSET" | loadkeys -u >/dev/null 2>&1 ||
      failed=1

    # If any of the commands above failed, the trap at the
    # top would set $failed to 1
    ( exit $failed )
    evaluate_retval

    exit $failed
    ;;

  *)
    echo "Usage:  ${0} {start}"
    exit 1
    ;;
esac

# End console
```

D.13. /etc/rc.d/init.d/localnet

```
#!/bin/sh
########################################################################
```

```
# Begin localnet
#
# Description : Loopback device
#
# Authors     : Gerard Beekmans  - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
##############################################################################

### BEGIN INIT INFO
# Provides:            localnet
# Required-Start:      mountvirtfs
# Should-Start:        modules
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:        0 6
# Short-Description:   Starts the local network.
# Description:         Sets the hostname of the machine and starts the
#                      loopback interface.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions
[ -r /etc/sysconfig/network ] && . /etc/sysconfig/network
[ -r /etc/hostname ] && HOSTNAME=`cat /etc/hostname`

case "${1}" in
   start)
      log_info_msg "Bringing up the loopback interface..."
      ip addr add 127.0.0.1/8 label lo dev lo
      ip link set lo up
      evaluate_retval

      log_info_msg "Setting hostname to ${HOSTNAME}..."
      hostname ${HOSTNAME}
      evaluate_retval
      ;;

   stop)
      log_info_msg "Bringing down the loopback interface..."
      ip link set lo down
      evaluate_retval
      ;;

   restart)
      ${0} stop
      sleep 1
      ${0} start
      ;;

   status)
      echo "Hostname is: $(hostname)"
      ip link show lo
```

```
    ;;

  *)
    echo "Usage: ${0} {start|stop|restart|status}"
    exit 1
    ;;
esac

exit 0

# End localnet
```

D.14. /etc/rc.d/init.d/sysctl

```
#!/bin/sh
########################################################################
# Begin sysctl
#
# Description : File uses /etc/sysctl.conf to set kernel runtime
#             parameters
#
# Authors     : Nathan Coulson (nathan@linuxfromscratch.org)
#             Matthew Burgress (matthew@linuxfromscratch.org)
#             DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            sysctl
# Required-Start:      mountvirtfs
# Should-Start:        console
# Required-Stop:
# Should-Stop:
# Default-Start:       S
# Default-Stop:
# Short-Description:   Makes changes to the proc filesystem
# Description:         Makes changes to the proc filesystem as defined in
#                      /etc/sysctl.conf.  See 'man sysctl(8)'.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    if [ -f "/etc/sysctl.conf" ]; then
      log_info_msg "Setting kernel runtime parameters..."
      sysctl -q -p
      evaluate_retval
    fi
    ;;

  status)
```

```
    sysctl -a
    ;;

  *)
    echo "Usage: ${0} {start|status}"
    exit 1
    ;;
esac

exit 0

# End sysctl
```

D.15. /etc/rc.d/init.d/sysklogd

```
#!/bin/sh
########################################################################
# Begin sysklogd
#
# Description : Sysklogd loader
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            $syslog
# Required-Start:      $first localnet
# Should-Start:
# Required-Stop:       $local_fs
# Should-Stop:         sendsignals
# Default-Start:       3 4 5
# Default-Stop:        0 1 2 6
# Short-Description:   Starts kernel and system log daemons.
# Description:         Starts kernel and system log daemons.
#                      /etc/fstab.
# X-LFS-Provided-By:   LFS
### END INIT INFO

# Note: sysklogd is not started in runlevel 2 due to possible
# remote logging configurations

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "Starting system log daemon..."
    parms=${SYSKLOGD_PARMS-'-m 0'}
    start_daemon /sbin/syslogd $parms
    evaluate_retval

    log_info_msg "Starting kernel log daemon..."
```

```
    start_daemon /sbin/klogd
    evaluate_retval
    ;;

  stop)
    log_info_msg "Stopping kernel log daemon..."
    killproc /sbin/klogd
    evaluate_retval

    log_info_msg "Stopping system log daemon..."
    killproc /sbin/syslogd
    evaluate_retval
    ;;

  reload)
    log_info_msg "Reloading system log daemon config file..."
    pid=`pidofproc syslogd`
    kill -HUP "${pid}"
    evaluate_retval
    ;;

  restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;

  status)
    statusproc /sbin/syslogd
    statusproc klogd
    ;;

  *)
    echo "Usage: ${0} {start|stop|reload|restart|status}"
    exit 1
    ;;
esac

exit 0

# End sysklogd
```

D.16. /etc/rc.d/init.d/network

```
#!/bin/sh
########################################################################
# Begin network
#
# Description : Network Control Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
```

```
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:            $network
# Required-Start:      $local_fs $syslog localnet swap
# Should-Start:        firewalld iptables nftables
# Required-Stop:       $local_fs $syslog localnet swap
# Should-Stop:         firewalld iptables nftables
# Default-Start:       3 4 5
# Default-Stop:        0 1 2 6
# Short-Description:   Starts and configures network interfaces.
# Description:         Starts and configures network interfaces.
# X-LFS-Provided-By:   LFS
### END INIT INFO

case "${1}" in
  start)
    # Start all network interfaces
    for file in /etc/sysconfig/ifconfig.*
    do
      interface=${file##*/ifconfig.}

      # Skip if $file is * (because nothing was found)
      if [ "${interface}" = "*" ]
      then
        continue
      fi

      /sbin/ifup ${interface}
    done
    ;;

  stop)
    # Unmount any network mounted file systems
     umount --all --force --types nfs,cifs,nfs4

    # Reverse list
    net_files=""
    for file in  /etc/sysconfig/ifconfig.*
    do
      net_files="${file} ${net_files}"
    done

    # Stop all network interfaces
    for file in ${net_files}
    do
      interface=${file##*/ifconfig.}

      # Skip if $file is * (because nothing was found)
      if [ "${interface}" = "*" ]
      then
        continue
      fi

      /sbin/ifdown ${interface}
```

```
    done
    ;;

  restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;

  *)
    echo "Usage: ${0} {start|stop|restart}"
    exit 1
    ;;
esac

exit 0

# End network
```

D.17. /etc/rc.d/init.d/sendsignals

```
#!/bin/sh
########################################################################
# Begin sendsignals
#
# Description : Sendsignals Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           sendsignals
# Required-Start:
# Should-Start:
# Required-Stop:      $local_fs swap localnet
# Should-Stop:
# Default-Start:
# Default-Stop:       0 6
# Short-Description:  Attempts to kill remaining processes.
# Description:        Attempts to kill remaining processes.
# X-LFS-Provided-By:  LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  stop)
    log_info_msg "Sending all processes the TERM signal..."
    killall5 -15
    error_value=${?}
```

```
    sleep ${KILLDELAY}

    if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
      log_success_msg
    else
      log_failure_msg
    fi

    log_info_msg "Sending all processes the KILL signal..."
    killall5 -9
    error_value=${?}

    sleep ${KILLDELAY}

    if [ "${error_value}" = 0 -o "${error_value}" = 2 ]; then
      log_success_msg
    else
      log_failure_msg
    fi
    ;;

  *)
    echo "Usage: ${0} {stop}"
    exit 1
    ;;

esac

exit 0

# End sendsignals
```

D.18. /etc/rc.d/init.d/reboot

```
#!/bin/sh
########################################################################
# Begin reboot
#
# Description : Reboot Scripts
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           reboot
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:      6
# Default-Stop:
```

```
# Short-Description:   Reboots the system.
# Description:         Reboots the System.
# X-LFS-Provided-By:   LFS
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  stop)
    log_info_msg "Restarting system..."
    reboot -d -f -i
    ;;

  *)
    echo "Usage: ${0} {stop}"
    exit 1
    ;;

esac

# End reboot
```

D.19. /etc/rc.d/init.d/halt

```
#!/bin/sh
########################################################################
# Begin halt
#
# Description : Halt Script
#
# Authors     : Gerard Beekmans - gerard@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

### BEGIN INIT INFO
# Provides:           halt
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:      0
# Default-Stop:
# Short-Description:   Halts the system.
# Description:         Halts the System.
# X-LFS-Provided-By:   LFS
### END INIT INFO

case "${1}" in
  stop)
    halt -d -f -i -p
    ;;
```

```
  *)
    echo "Usage: {stop}"
    exit 1
    ;;
esac

# End halt
```

D.20. /etc/rc.d/init.d/template

```
#!/bin/sh
########################################################################
# Begin scriptname
#
# Description :
#
# Authors     :
#
# Version     : LFS x.x
#
# Notes       :
#
########################################################################

### BEGIN INIT INFO
# Provides:           template
# Required-Start:
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:
# Default-Stop:
# Short-Description:
# Description:
# X-LFS-Provided-By:
### END INIT INFO

. /lib/lsb/init-functions

case "${1}" in
  start)
    log_info_msg "Starting..."
    start_daemon fully_qualified_path
    ;;

  stop)
    log_info_msg "Stopping..."
    killproc fully_qualified_path
    ;;

  restart)
    ${0} stop
    sleep 1
    ${0} start
    ;;
```

```
  *)
    echo "Usage: ${0} {start|stop|restart}"
    exit 1
    ;;
esac

exit 0

# End scriptname
```

D.21. /etc/sysconfig/modules

```
#############################################################################
# Begin /etc/sysconfig/modules
#
# Description : Module auto-loading configuration
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#         <module> [<arg1> <arg2> ...]
#
# Each module should be on its own line, and any options that you want
# passed to the module should follow it.  The line deliminator is either
# a space or a tab.
#############################################################################

# End /etc/sysconfig/modules
```

D.22. /etc/sysconfig/createfiles

```
#############################################################################
# Begin /etc/sysconfig/createfiles
#
# Description : Createfiles script config file
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : The syntax of this file is as follows:
#         if type is equal to "file" or "dir"
#          <filename> <type> <permissions> <user> <group>
#         if type is equal to "dev"
#          <filename> <type> <permissions> <user> <group> <devtype>
#            <major> <minor>
#
#         <filename> is the name of the file which is to be created
#         <type> is either file, dir, or dev.
#              file creates a new file
#              dir creates a new directory
#              dev creates a new device
#         <devtype> is either block, char or pipe
```

```
#             block creates a block device
#             char creates a character deivce
#             pipe creates a pipe, this will ignore the <major> and
#          <minor> fields
#        <major> and <minor> are the major and minor numbers used for
#     the device.
#########################################################################

# End /etc/sysconfig/createfiles
```

D.23. /etc/sysconfig/udev-retry

```
#########################################################################
# Begin /etc/sysconfig/udev_retry
#
# Description : udev_retry script configuration
#
# Authors     :
#
# Version     : 00.00
#
# Notes       : Each subsystem that may need to be re-triggered after mountfs
#             runs should be listed in this file.  Probable subsystems to be
#             listed here are rtc (due to /var/lib/hwclock/adjtime) and sound
#             (due to both /var/lib/alsa/asound.state and /usr/sbin/alsactl).
#             Entries are whitespace-separated.
#########################################################################

rtc

# End /etc/sysconfig/udev_retry
```

D.24. /sbin/ifup

```
#!/bin/sh
#########################################################################
# Begin /sbin/ifup
#
# Description : Interface Up
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#             Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#             DJ Lucas - dj@linuxfromscratch.org
#
# Version     : LFS 7.7
#
# Notes       : The IFCONFIG variable is passed to the SERVICE script
#             in the /lib/services directory, to indicate what file the
#             service should source to get interface specifications.
#
#########################################################################

up()
{
```

```
  log_info_msg "Bringing up the ${1} interface..."

  if ip link show $1 > /dev/null 2>&1; then
    link_status=`ip link show $1`

    if [ -n "${link_status}" ]; then
      if ! echo "${link_status}" | grep -q UP; then
        ip link set $1 up
      fi
    fi

  else
    log_failure_msg "Interface ${IFACE} doesn't exist."
    exit 1
  fi

  evaluate_retval
}

RELEASE="7.7"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
VERSTR="LFS ifup, version ${RELEASE}"

while [ $# -gt 0 ]; do
   case "$1" in
     --help | -h)     help="y"; break ;;

     --version | -V)  echo "${VERSTR}"; exit 0 ;;

     -*)              echo "ifup: ${1}: invalid option" >&2
                      echo "${USAGE}" >& 2
                      exit 2 ;;

     *)               break ;;
   esac
done

if [ -n "$help" ]; then
   echo "${VERSTR}"
   echo "${USAGE}"
   echo
   cat << HERE_EOF
ifup is used to bring up a network interface.  The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
   exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%""~""}" ] || exit 0

. /lib/lsb/init-functions
```

```
if [ ! -r "${file}" ]; then
   log_failure_msg "Unable to bring up ${1} interface! ${file} is missing or cannot be accessed."
   exit 1
fi

.  $file

if [ "$IFACE" = "" ]; then
   log_failure_msg "Unable to bring up ${1} interface! ${file} does not define an interface [IFACE]."
   exit 1
fi

# Do not process this service if started by boot, and ONBOOT
# is not set to yes
if [ "${IN_BOOT}" = "1" -a "${ONBOOT}" != "yes" ]; then
   exit 0
fi

# Bring up the interface
if [ "$VIRTINT" != "yes" ]; then
   up ${IFACE}
fi

for S in ${SERVICE}; do
  if [ ! -x "/lib/services/${S}" ]; then
    MSG="₩nUnable to process ${file}.  Either "
    MSG="${MSG}the SERVICE '${S} was not present "
    MSG="${MSG}or cannot be executed."
    log_failure_msg "$MSG"
    exit 1
  fi
done

if [ "${SERVICE}" = "wpa" ]; then log_success_msg; fi

# Create/configure the interface
for S in ${SERVICE}; do
  IFCONFIG=${file} /lib/services/${S} ${IFACE} up
done

# Set link up virtual interfaces
if [ "${VIRTINT}" == "yes" ]; then
   up ${IFACE}
fi

# Bring up any additional interface components
for I in $INTERFACE_COMPONENTS; do up $I; done

# Set MTU if requested. Check if MTU has a "good" value.
if test -n "${MTU}"; then
   if [[ ${MTU} =~ ^[0-9]+$ ]] && [[ $MTU -ge 68 ]] ; then
      for I in $IFACE $INTERFACE_COMPONENTS; do
        ip link set dev $I mtu $MTU;
      done
   else
      log_info_msg2 "Invalid MTU $MTU"
```

```
  fi
fi

# Set the route default gateway if requested
if [ -n "${GATEWAY}" ]; then
  if ip route | grep -q default; then
    log_warning_msg "Gateway already setup; skipping."
  else
    log_info_msg "Adding default gateway ${GATEWAY} to the ${IFACE} interface..."
    ip route add default via ${GATEWAY} dev ${IFACE}
    evaluate_retval
  fi
fi

# End /sbin/ifup
```

D.25. /sbin/ifdown

```
#!/bin/bash
########################################################################
# Begin /sbin/ifdown
#
# Description : Interface Down
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
# Notes       : the IFCONFIG variable is passed to the scripts found
#               in the /lib/services directory, to indicate what file the
#               service should source to get interface specifications.
#
########################################################################

RELEASE="7.0"

USAGE="Usage: $0 [ -hV ] [--help] [--version] interface"
VERSTR="LFS ifdown, version ${RELEASE}"

while [ $# -gt 0 ]; do
  case "$1" in
    --help | -h)     help="y"; break ;;

    --version | -V)  echo "${VERSTR}"; exit 0 ;;

    -*)              echo "ifup: ${1}: invalid option" >&2
                     echo "${USAGE}" >& 2
                     exit 2 ;;

    *)               break ;;
  esac
done

if [ -n "$help" ]; then
```

```
    echo "${VERSTR}"
    echo "${USAGE}"
    echo
    cat << HERE_EOF
ifdown is used to bring down a network interface.  The interface
parameter, e.g. eth0 or eth0:2, must match the trailing part of the
interface specifications file, e.g. /etc/sysconfig/ifconfig.eth0:2.

HERE_EOF
    exit 0
fi

file=/etc/sysconfig/ifconfig.${1}

# Skip backup files
[ "${file}" = "${file%""~""}" ] || exit 0

. /lib/lsb/init-functions

if [ ! -r "${file}" ]; then
    log_warning_msg "${file} is missing or cannot be accessed."
    exit 1
fi

. ${file}

if [ "$IFACE" = "" ]; then
    log_failure_msg "${file} does not define an interface [IFACE]."
    exit 1
fi

# We only need to first service to bring down the interface
S=`echo ${SERVICE} | cut -f1 -d" "`

if ip link show ${IFACE} > /dev/null 2>&1; then
    if [ -n "${S}" -a -x "/lib/services/${S}" ]; then
      IFCONFIG=${file} /lib/services/${S} ${IFACE} down
    else
      MSG="Unable to process ${file}.  Either "
      MSG="${MSG}the SERVICE variable was not set "
      MSG="${MSG}or the specified service cannot be executed."
      log_failure_msg "$MSG"
      exit 1
   fi
else
    log_warning_msg "Interface ${1} doesn't exist."
fi

# Leave the interface up if there are additional interfaces in the device
link_status=`ip link show ${IFACE} 2>/dev/null`

if [ -n "${link_status}" ]; then
    if [ "$(echo "${link_status}" | grep UP)" != "" ]; then
       if [ "$(ip addr show ${IFACE} | grep 'inet ')" == ""  ]; then
          log_info_msg "Bringing down the ${IFACE} interface..."
          ip link set ${IFACE} down
          evaluate_retval
```

```
    fi
  fi
fi

# End /sbin/ifdown
```

D.26. /lib/services/ipv4-static

```
#!/bin/sh
########################################################################
# Begin /lib/services/ipv4-static
#
# Description : IPV4 Static Boot Script
#
# Authors     : Nathan Coulson - nathan@linuxfromscratch.org
#               Kevin P. Fleming - kpfleming@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

. /lib/lsb/init-functions
. ${IFCONFIG}

if [ -z "${IP}" ]; then
  log_failure_msg "₩nIP variable missing from ${IFCONFIG}, cannot continue."
  exit 1
fi

if [ -z "${PREFIX}" -a -z "${PEER}" ]; then
  log_warning_msg "₩nPREFIX variable missing from ${IFCONFIG}, assuming 24."
  PREFIX=24
  args="${args} ${IP}/${PREFIX}"

elif [ -n "${PREFIX}" -a -n "${PEER}" ]; then
  log_failure_msg "₩nPREFIX and PEER both specified in ${IFCONFIG}, cannot continue."
  exit 1

elif [ -n "${PREFIX}" ]; then
  args="${args} ${IP}/${PREFIX}"

elif [ -n "${PEER}" ]; then
  args="${args} ${IP} peer ${PEER}"
fi

if [ -n "${LABEL}" ]; then
  args="${args} label ${LABEL}"
fi

if [ -n "${BROADCAST}" ]; then
  args="${args} broadcast ${BROADCAST}"
fi

case "${2}" in
  up)
```

```
    if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" = "" ]; then
      log_info_msg "Adding IPv4 address ${IP} to the ${1} interface..."
      ip addr add ${args} dev ${1}
      evaluate_retval
    else
      log_warning_msg "Cannot add IPv4 address ${IP} to ${1}.  Already present."
    fi
  ;;

  down)
    if [ "$(ip addr show ${1} 2>/dev/null | grep ${IP}/)" != "" ]; then
      log_info_msg "Removing IPv4 address ${IP} from the ${1} interface..."
      ip addr del ${args} dev ${1}
      evaluate_retval
    fi

    if [ -n "${GATEWAY}" ]; then
      # Only remove the gateway if there are no remaining ipv4 addresses
      if [ "$(ip addr show ${1} 2>/dev/null | grep 'inet ')" != "" ]; then
        log_info_msg "Removing default gateway..."
        ip route del default
        evaluate_retval
      fi
    fi
  ;;

  *)
    echo "Usage: ${0} [interface] {up|down}"
    exit 1
  ;;
esac

# End /lib/services/ipv4-static
```

D.27. /lib/services/ipv4-static-route

```
#!/bin/sh
########################################################################
# Begin /lib/services/ipv4-static-route
#
# Description : IPV4 Static Route Script
#
# Authors     : Kevin P. Fleming - kpfleming@linuxfromscratch.org
#               DJ Lucas - dj@linuxfromscratch.org
# Update      : Bruce Dubbs - bdubbs@linuxfromscratch.org
#
# Version     : LFS 7.0
#
########################################################################

. /lib/lsb/init-functions
. ${IFCONFIG}

case "${TYPE}" in
  ("" | "network")
    need_ip=1
```

```
    need_gateway=1
  ;;

  ("default")
    need_gateway=1
    args="${args} default"
    desc="default"
  ;;

  ("host")
    need_ip=1
  ;;

  ("unreachable")
    need_ip=1
    args="${args} unreachable"
    desc="unreachable "
  ;;

  (*)
    log_failure_msg "Unknown route type (${TYPE}) in ${IFCONFIG}, cannot continue."
    exit 1
  ;;
esac

if [ -n "${GATEWAY}" ]; then
  MSG="The GATEWAY variable cannot be set in ${IFCONFIG} for static routes.₩n"
  log_failure_msg "$MSG Use STATIC_GATEWAY only, cannot continue"
  exit 1
fi

if [ -n "${need_ip}" ]; then
  if [ -z "${IP}" ]; then
    log_failure_msg "IP variable missing from ${IFCONFIG}, cannot continue."
    exit 1
  fi

  if [ -z "${PREFIX}" ]; then
    log_failure_msg "PREFIX variable missing from ${IFCONFIG}, cannot continue."
    exit 1
  fi

  args="${args} ${IP}/${PREFIX}"
  desc="${desc}${IP}/${PREFIX}"
fi

if [ -n "${need_gateway}" ]; then
  if [ -z "${STATIC_GATEWAY}" ]; then
    log_failure_msg "STATIC_GATEWAY variable missing from ${IFCONFIG}, cannot continue."
    exit 1
  fi
  args="${args} via ${STATIC_GATEWAY}"
fi

if [ -n "${SOURCE}" ]; then
      args="${args} src ${SOURCE}"
fi
```

```
case "${2}" in
  up)
    log_info_msg "Adding '${desc}' route to the ${1} interface..."
    ip route add ${args} dev ${1}
    evaluate_retval
  ;;

  down)
    log_info_msg "Removing '${desc}' route from the ${1} interface..."
    ip route del ${args} dev ${1}
    evaluate_retval
  ;;

  *)
    echo "Usage: ${0} [interface] {up|down}"
    exit 1
  ;;
esac

# End /lib/services/ipv4-static-route
```

부록 E. Udev 구성 규칙

이 부록의 규칙은 편의를 위해 열거되어 있다. 일반적으로 설치는 6.78절. “Eudev-3.2.9”의 지침을 통해 이루어진다.

E.1. 55-lfs.rules

```
# /etc/udev/rules.d/55-lfs.rules: Rule definitions for LFS.

# Core kernel devices

# This causes the system clock to be set as soon as /dev/rtc becomes available.
SUBSYSTEM=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"
KERNEL=="rtc", ACTION=="add", MODE="0644", RUN+="/etc/rc.d/init.d/setclock start"

# Comms devices

KERNEL=="ippp[0-9]*",       GROUP="dialout"
KERNEL=="isdn[0-9]*",       GROUP="dialout"
KERNEL=="isdnctrl[0-9]*",   GROUP="dialout"
KERNEL=="dcbri[0-9]*",      GROUP="dialout"
```

부록 F. LFS 라이센스

F.1. Creative Commons License

f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

g. "License Elements" means the following high-level license attributes as selected by Licensor and indicated in the title of this License: Attribution, Noncommercial, ShareAlike.

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;

   b. to create and reproduce Derivative Works;

   c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;

   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission Derivative Works;

   The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Sections 4(e) and 4(f).

4. Restrictions.The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

   a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any reference to such Licensor or the Original Author, as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any reference to such Licensor or the Original Author, as requested.

   b. You may distribute, publicly display, publicly perform, or publicly digitally perform a Derivative Work only under the terms of this License, a later version of this License with the same License Elements as this License, or a Creative Commons iCommons license that contains the same

License Elements as this License (e.g. Attribution-NonCommercial-ShareAlike 2.0 Japan). You must include a copy of, or the Uniform Resource Identifier for, this License or other license specified in the previous sentence with every copy or phonorecord of each Derivative Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Derivative Works that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder, and You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Derivative Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Derivative Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Derivative Work itself to be made subject to the terms of this License.

c. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

d. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and give the Original Author credit reasonable to the medium or means You are utilizing by conveying the name (or pseudonym if applicable) of the Original Author if supplied; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

e. For the avoidance of doubt, where the Work is a musical composition:

   i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

   ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation. 6. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance

(e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

5. Representations, Warranties and Disclaimer

   UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

   a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

   b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

   a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

   b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

중요

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, neither party will use the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time.

Creative Commons may be contacted at http://creativecommons.org/.

F.2. The MIT License

색인

패키지들

프로그램들

라이브러리들

스크립트들

기타