Table of Contents

환경설정 및 재부팅

0. Overview

이 문서는 어떠한 linux distro를 만들어낼 수 있는 Linux from scratch(이하 LFS)를 제작하는 문서입니다. 개인의 취향과 최적화를 부합하기 위해서는 LFS가 가장 적합합니다. 저희는 패키지를 관리하고 설치하는 패키지 매니지먼트 또한 무게를 최대한 줄여가며 만들기 때문에, 작지만 안전한 임베디드 리눅스 시스템을 LFS로 구축할 것입니다.

버전 9.1 systemd LFS을 사용하였고 2020년 3월 1일에 발행한 자료로 최신인 편에 속합니다. 다음 목차는 1999년 이후로 쓰여진 모든 LFS 문서와 일치합니다. 그렇기 때문에 타 LFS 공식 문서와 목차를 통일시켰고, 만약 컴파일이나 진행을 하면서 어려움이 생길 시 이 문서로 이해가 부족하실 때엔 해당 목차의 번호를 기억하여 구글이나 LFS 관련 커뮤니티에 검색하시면 많은 정보를 얻으실 수 있습니다.

<* 단 7장 이후로는 저희가 debian from scratch라는 주제를 다룬 깃허브를 참고했으며, 리눅스 빌드 과정을 재편집하여 진행하였습니다. 그렇기 때문에, 7장부터는 문서를 더 집중해서 봐주시면 감사하겠습니다. debian from scratch 를 다룬 깃허브를 밑에 같이 첨부하겠습니다. >**

깃허브 링크 : https://github.com/scottwilliambeasley/debian-from-scratch

편의를 위해 호스트 시스템은 Ubuntu-18.04 LTS 버전을 사용하였습니다.

1. LFS를 어떻게 만들 것인가?

챕터 2~3에서는 파티션을 나누고 필요한 호스트 환경을 구축합니다.

챕터 4~5에서는 유저를 root에서 LFS로 바꿔서 dpkg설치에 필요한 임시 툴들을 설치합니다.

챕터 6~10은 chroot environment에서 dpkg, apt 설치 및 최종 빌드시스템에 들어갈 패키지를 설치합니다.

챕터 11은 시스템을 재부팅하여 최종 빌드한 시스템으로 재부팅합니다.

2. 호스트 시스템 준비하기

이 장에서는 LFS를 구축하기 위한 호스트 시스템의 tool와 디펜던시를 확인하고 필요한 경우 설치합니다. 그 다음, LFS을 호스팅할 파티션을 구축합니다. 그 다음 파티션 자체에 파일 시스템을 제작하고 마운트할 것입니다.

호스트 시스템 요구사항

해당 방법은 18.04 LTS 에서 진행하였습니다.

먼저 새로운 우분투 리눅스에서 작업을 하기 전에 LFS 설치에 필요한 디펜던시를 우선적으로 확인합니다.

```
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
```

이를 확인하고, found가 되지 않은 커맨드들이 있다면 원활한 진행을 위해 설치를 하고 넘어가야 합니다. 예를 들어,

`version-check.sh: line 51: g++: command not found` 라 나오면 `sudo apt-get install g++` 을 하는 등 모든 디펜던시가 설치가 됨을 확인하고 넘어갑니다.

18.04 LTS 를 베이스로 설치하자마자 저 명령어를 사용했을 시,

```
sudo apt-get install bison gawk gcc g++ make texinfo

sudo ln -sf bash /bin/bash
```

를 진행했더니 모든 디펜던시가 설치되었습니다. 각 리눅스 distro 혹은 버전 및 설치 환경마다 상황이 다르기 때문에 디펜던시 유무 확인을 해보시고 진행을 하시길 바랍니다.

새로운 파티션 생성

새롭게 만들어줄 LFS가 들어설 파티션을 생성합니다. 최소한 5GB 이상으로 공간을 확보하면 좋습니다. 추가로 디스크를 부여해 해당 디스크를 파티션으로 사용합니다. (가상머신에서는 하드디스크를 임의로 부여할 수 있습니다. 리눅스 환경에서도 부여가 가능합니다.)

파티션에 파일시스템 생성하기

`sudo fdisk -l` : 현재 디스크 확인

`sudo cfdisk /dev/***` LFS를 이식할 디스크에 파티션 설정

`<xxx>` : bootable, linux

`<yyy>` : for linux swap/ solaris

로 설정을 마친 후 `sudo fdisk -l`을 하여 제대로 설정이 됐는지 확인합니다. 그리고 제대로 설정이 되어 있으면 파티션한 하드디스크를 포맷하고 스왑지정합니다.

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

`sudo mkswap /dev/<yyy>`

(파티션 분할이 익숙치 않으시다면 다음 링크를 참고해주세요.)

https://www.youtube.com/watch?v=YQxi3S6eSIQ

환경변수 설정

$LFS라는 변수를 환경변수로 설정해 향후 설치할 때도 사용합니다.

`export LFS=/mnt/lfs` : LFS를 환경변수로 설정합니다.

`echo $LFS` : 설정이 잘 되었는지 확인합니다.

새로운 파티션 마운트 하기

직전에 만들었던 새로운 파티션을 마운트 하는 작업입니다. 작업하기 앞서서 `su` 를 이용해 유저를 root로 바꿔 진행하면 수월합니다. 이 다음부터는 sudo를 제외하고 코드 진행하겠습니다.

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

를 해당되는 LFS 파티션으로 대체해서 사용하세요.

만약 swap 파티션을 사용했다면,

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

을 진행하시면 됩니다. (저는 하나의 LFS 파티션과 하나의 Swap파티션만 사용했습니다.)

<* 파티션을 생성하고 부여하는 것은 자유롭게 진행하면 됩니다. LFS, usr, swap 이렇게 3개의 파티션을 주어도 되며 LFS, swap 2개의 파티션을 부여해도 무리없습니다. >**

여기까지 진행했으면 이제는 LFS에 필요한 패키지와 패치 파일을 설치하는 과정입니다 !

3. 필요한 패키지와 패치 파일

이 장에서는 LFS를 구축하기 위해 꼭 다운로드를 해야하는 패키지 목록을 가져오는 작업을 할 것입니다. 나열된 버전에 꼭 맞는 패키지를 사용해야 하기 때문에 앞선 overview에서 말씀을 드렸듯, 9.1 systemd에 맞는 패키지를 가져와야 설치가 원활합니다.

필요한 파일 설치

root로 계정을 바꾸시고, 소스파일을 풀어낼 파일을 $LFS 내에 만듭니다.

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

그리고 http://www.linuxfromscratch.org/lfs/downloads/9.1-systemd/wget-list 에 있는 리스트를 wget 명령어를 이용하여 다운 받습니다.

```
wget http://www.linuxfromscratch.org/lfs/downloads/9.1-systemd/wget-list
```

그 다음에 wget-list 안에 담겨 있는 패키지 모음을 $LFS/sources 안에 담습니다.

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

해당 패키지들이 올바른 패키지임을 체크할 수 있는 별도의 파일 md5sum이 존재합니다. 해당 파일들을 다운로드합니다.

```
wget http://www.linuxfromscratch.org/lfs/downloads/9.1-systemd/md5sums
```

올바른 패키지임을 확인합니다.

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

여기까지 진행이 되었으면 ls 하여 sources 파일 내에 패키지들이 올바르게 설치되었는지 확인합니다.

4. 마지막 준비

이 장에서는 temporary tools를 구축하기 위한 몇 가지 추가 작업을 수행할 것입니다.

필요한 디렉토리 생성과 환경설정하기

임시 시스템을 구축하기 전에 몇가지 추가 작업을 진행하는 과정입니다. $LFS temporary tool 을 설치하고 해당 디렉토리에 권한이 없는 사용자를 추가하여 위험을 줄이고 해당 사용자에 맞게 빌드 환경을 커스터마이징하는 단계입니다.

이를 실행하기 위해 필요한 디렉토리를 만듭니다.

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

그 다음, /tools 호스트 시스템에 심볼릭 링크를 만듭니다. 이 과정을 통해서 새로 생긴 디렉토리를 가리킵니다.

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

이제 사용자를 추가합니다. 권한이 없는 사용자를 지정하여 하는 것이 안전하기 때문에 저희도 권한이 없는 사용자를 추가해서 진행합니다. 이름은 편하게 lfs로 부여합니다. root 권한으로 이를 진행합니다.

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

디렉토리 소유자를 lfs로 지정해서 $LFS/tools 에 대한 전체 엑세스 권한을 lfs에 부여합니다. 또한 사용자의 소유권을 lfs에게 추가로 제공합니다.

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

이제 로그인합니다. 그러면 프롬프트가 root에서 lfs로 바뀔 것입니다.

```
su - lfs
```

환경설정을 하는 단계입니다. lfs에 로그인을 한 상태에서 새로운 환경에서 그 전과 유사한 환경으로 설정합니다.

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

```
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
```

마지막으로 temporary tool을 빌드하기 위한 환경을 준비하기 위해 방금 만든 것을 sourcing합니다.

```
source ~/.bash_profile
```

5. 임시 시스템 구축

이제 최종 LFS를 구축하기 위해 충분한 tool들을 구축하는 단계입니다. 필요한 temporary tool만 올려 리눅스를 제작합니다.

먼저 환경변수인 $LFS가 lfs에서도 잘 설정되어 있는지 확인합니다.

```
echo $LFS
```

`/mnt/lfs` 가 잘 나오게 되면, 이제 필요한 시스템을 구축하도록 합니다 !

```
cd $LFS/sources
```

이제 소스코드가 있는 곳에 들어가 작업을 준비합니다. 진행을 하는 유저는 root가 아닌 lfs입니다.

주의사항

<* 앞으로 temporary tools을 구축하기 위해 필수 패키지들을 설치할 예정입니다. 과정에서 예기치 않은 오류가 발생할 수도 있으니 아래 적어진 명령어를 그대로 쳐주시거나 복사해서 입력해주시기 바랍니다.**

가장 많이 나올 수 있는 에러는 " Error : ~ " 입니다. 이러한 메시지가 나오게 되면 진행해 온 과정에서 문제가 발생했을 가능성이 있어 처음부터 차분히 다시 진행하는 것도 방법입니다.

"Nothing to be done : ~" 이러한 메시지가 make 직후 발생하는 경우가 많은데 문제가 되지 않음을 확인하였습니다.

이는 가이드 전체에 적용되는 주의사항이니 이를 기억하고 진행해주시면 감사하겠습니다. *>**

Binutils-2.34

해당 패키지는 어셈블리어 및 여러 파일에 대한 파일처리를 위한 도구가 포함되어 있습니다. 이 패키지는 LFS 시스템의 패키지를 컴파일하는 용도이기 때문에 꼭 필요합니다.

해당 패키지를 설치하기 위해서 $LFS/sources 로 이동하여 디렉토리 안에 binutils-2.34.tar.xz 파일의 압축을 풀고 생성된 디렉토리로 이동해서 작업합니다. 그리고, 원활한 패키지 설치를 진행하기 위해 build 만을 진행하는 디렉토리를 하나 생성합니다.

```
tar -xvf binutils-2.34.tar.xz
cd binutils-2.34
mkdir -v build
cd       build
```

이제 컴파일을 위해 binutils의 구성 옵션을 조정합니다.

<* 모든 패키지를 설치하게 되면, 무거운 리눅스 시스템이 완성이 될 것입니다. 사용하고자 하는 패키지의 범위만 설치하기 위해 진행하는 과정입니다. 이는 모든 소스파일 설치를 하는 과정에 동일합니다.>**

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

구성 옵션의 의미: --prefix=/tools : binutils 프로그램을 설치하도록 구성 스크립트에 지시합니다. --with-sysroot=$LFS : 크로스컴파일의경우, 필요에따라LFS를 찾도록 합니다. --with-lib-path=/tools/lib : 환경변수 설정을 통해 경로를 지정합니다. --target=$LFS_TGT : LFS_TGT 변수의 시스템이 실제 스크립트에서 반환하는 값이 다르기 때문에 이를 조정합니다. --disable-nls : temporary tools에는 필요하지 않는 nls를 비활성화합니다. --disable-werror : 컴파일 중 경고가 발생되더라도 진행이 되게끔 합니다.

패키지 컴파일을 진행합니다 .

```
make
```

x86_64에서 빌드하는 경우 심볼릭 링크를 제작해둡니다.

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

패키지를 설치합니다.

```
make install
```

사용을 다하면 삭제합니다. 압축을 사용하여 풀어낸 파일만 삭제하는 것입니다.

```
cd ../../
rm -Rf binutils-2.35
```

GCC-9.2.0

gnu 컴파일러입니다. c 및 c++ 컴파일러 등 여러 컴파일러가 존재하기 때문에 필수입니다.

해당 패키지를 설치하기 위해 gcc-9.2.0.tar.xz 파일의 압축을 풀고 해당 디렉토리로 이동합니다.

< * 밑 라인 2줄의 과정은 5장 내내 동일하기 때문에 이 다음부터는 생략하겠습니다. >**

```
tar -xvf gcc-10.2.0.tar.xz
cd gcc-10.2.0
```

gcc에는 gmp, mpfr, mpc 패키지가 필요합니다. 이들을 각각 소스코드로 풀고, gcc 빌드 절차에 자동으로 사용할 수 있게 설정합니다.

```
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
```

그 다음, gcc의 기본 동적 링커의 위치를 변경합니다.

```
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
```

마지막으로 x86_64 호스트의 경우는 라이브러리의 기본 디렉토리 이름을 'lib'로 설정합니다.

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

gcc 문서 전용 빌드 디렉토리를 생성합니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

```
../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++
```

구성 옵션의 의미: --with-glibc-version=2.11 : 이 옵션은 패키지가 호스트의 glibc 버전과 호환이 되는지를 확인합니다. --with-newlib : 아직 C 라이브러리를 사용할 수는 없기 때문에 libgcc 를 빌드할 때 문제가 되지 않게 컴파일을 진행하게 합니다. --without-headers : 크로스 컴파일러를 제작할 때 원래 타겟 시스템에는 헤더가 요구되지만 저희는 헤더가 요구되지 않기 때문에 무시하는 옵션입니다. --disable-shared : 내부 라이브러리가 정적으로 연결이 되도록 합니다. --disable-decimal-float, --disable-threads, --disable-libatomic, --disable-libgomp, --disable-libquadmath, --disable-libssp, --disable-libvtv, --disable-libstdcxx : temporary tools에는 필요가 없는 작업들이라 제외합니다. --disable-multilib : x86_64 에서는 multilib을 지원하지 않기 때문에 제외합니다. --enable-languages=c,c++ : C 및 C++ 컴파일러만 빌드되도록 하는 옵션입니다.

컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

동일하게 해당 디렉토리를 삭제합니다. tar로 압축을 푼 파일만 삭제하는 것입니다.

< * 밑 라인 2줄의 과정은 5장 내내 동일하기 때문에 이 다음부터는 생략하겠습니다. >**

```
cd ../../
rm -Rf gcc-10.2.0
```

linux-5.5.3 api header

이 소스파일은 linux 커널이 기본적으로 가져야 하는 api header에 관한 것입니다. 필수입니다.

패키지에 오래된 파일이 있는 지 먼저 확인합니다. (앞서 먼저 파일을 tar를 풀고 해당 파일로 들어와야 합니다.)

```
make mrproper
```

소스에서 사용자가 볼 수 있는 커널 헤더를 추출하는 작업입니다. 헤더는 먼저 배치된 ./usr 다음 필요한 위치에 놓이게 됩니다.

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

해당 파일은 source 까지 나가 삭제합니다.

Glibc-2.31

이 소스파일에는 기본 C 라이브러리가 포함되어 있습니다. 필수입니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

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

구성 옵션의 의미: --enable-kernel=3.2 : 3.2 이상 linux 커널을 지원하는 라이브러리에서만 컴파일을 하도록 지시하는 옵션입니다. --with-headers=/tools/include : tools/ 디렉토리 내 최근에 설치된 헤더에 대해서만 컴파일을 할 수 있도록 하는 옵션입니다. 패키지를 컴파일 합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Libstdc++ from gcc-9.2.0

Libstdc++는 표준 c++라이브러리입니다. c++를 컴파일하는데 필요하지만 앞서 진행했던 gcc-9.2.0은 /tools 에 대한 컴파일이기 때문에 동시에 하지 못하였습니다. 지금 표준 c++라이브러리를 설치하기 위해 gcc-9.2.0을 다시 tar 압축을 풀어 진입합니다.

빌드 및 패키지 구성을 위한 디렉토리를 만듭니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

```
../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
```

구성 옵션의 의미: --disable-libstdcxx-threads : C++ 스레드 라이브러리를 지원하지 않기 때문에 이를 빌드하지 않도록 합니다. --disable-libstdcxx-pch : 미리 컴파일을 하는 것을 방지합니다. --with-gxx-include-dir=/tools/$LFS_TGT/include/c++/9.2.0 : C++ 컴파일러가 표준을 포함하는 파일을 검색하는 위치가 됩니다.

libstdc++를 컴파일합니다.

```
make
```

설치합니다.

```
make install
```

Binutils-2.34 - pass 2

binutils 패키지에는 앞서 진행한 디펜던시를 제외하고도 링커, 어셈블러 등 개체 파일 처리를 진행하는 도구가 있습니다. 이에 대해 설치합니다. 필수입니다.

빌드 및 패키지 구성을 위한 디렉토리를 만듭니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

```
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
```

구성 옵션의 의미: CC=$LFS_TGT-gcc$ AR=$LFS_TGT-ar RANLIB=$LFS_TGT-ranlib : 실제 binutils의 기본 빌드를 진행하기 때문에 이러한 변수들을 설정하여 호스트 시스템의 도구를 사용하는 대신 크로스 컴파일러의 도구를 사용하게끔 합니다.

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

향후 6장에서 dpkg에 chroot진입을 시도합니다. 이를 위해 링커를 미리 준비합니다.

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

Gcc-9.2.0 - pass 2

gcc 패키지에는 앞서 진행한 것과 함께 c, c++ 컴파일러를 포함하는 GNU 컴파일러 모음이 있습니다. 필수입니다. 가장 오래 걸리는 컴파일입니다.

앞서 진행한 gcc 설치에서 몇가지 내부시스템 헤더를 설치했습니다. 그 중 하나인 limits.h가 다음 진행될 빌드에서 limits.h헤더에 포함됩니다. 이것은 임시 libc를 빌드하기엔 적합했지만 gcc 빌드 전반적으로 전체 버전 헤더가 필요합니다. 일반적인 환경에서 gcc 빌드 시스템을 수행하는 것과 동일한 명령을 사용하여 내부 헤더를 글로벌 버전으로 제작합니다.

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

gcc의 기본 동적 링커 위치를 /tools 로 변경합니다.

```
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 64 비트 라이브러리의 기본 디렉토리 이름을 "lib"로 변경합니다.

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

gcc 첫번째 빌드와 마찬가지로 gmp, mpfr, mpc 패키지가 필요합니다. 설치에 용이한 이름으로 변경해서 사용합니다.

```
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
```

앞서 glibc-2.31에서 진행했을 때 생길 수 있는 취약점을 미리 조정합니다.

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

빌드 및 패키지 구성을 위한 디렉토리를 만듭니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

```
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
```

구성 옵션의 의미: --enable-languages=c,c++ : C, C++ 컴파일러가 빌드됩니다. --disable-libstdcxx-pch : 미리 컴파일을 한 헤더의 빌드를 방지합니다. --disable-bootstrap : gcc 의 기본값은 boostrap을 빌드하는 것입니다. 이는 gcc를 여러 번 컴파일하는 특징이 있습니다. LFS는 매번 bootstrap 방식의 빌드를 필요로 하지 않습니다. 그래서 비활성화합니다.

컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

심볼릭 링크를 걸어 마무리합니다.

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

Ncurses-6.2

이 패키지는 character screen에 대해 터미널의 독립적 처리를 위한 라이브러리를 포함합니다. 향후 dpkg을 설치할 때 필요합니다. 필수입니다.

구성 중에 gawk가 먼저 발견되었는지 확인합니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

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

구성 옵션의 의미: --without-ada : ada 컴파일러는 향후 진입할 chroot 환경에서는 사용이 불가능하기 때문에 비활성화합니다. --enable-overwrite : 다른 패키지가 ncurses 헤더를 쉽게 찾을 수 있도록 /tools/include 대신 /tools/include/ncurses 로 overwrite 합니다.

패키지를 컴파일합니다.

```
make
```

설치하고 링크를 걸어 마무리합니다.

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

Bash-5.0

향후 컴파일 및 dpkg 빌드에 있어서 필수적입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

쉘에 sh를 사용하는 프로그램에 대한 링크를 걸어주면 됩니다.

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

Coreutils-8.31

기본 시스템 및 세팅을 보이는 데에 필요한 패키지입니다. 향후 dpkg 설치에 필수적입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

구성 옵션의 의미: --enable-install-program=hostname : 호스트 이름을 바이너리로 빌드 및 설치를 할 수 있게 하는 명령어입니다. 향후 perl 테스트 에서 필요합니다.

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Diffutils-3.7

파일 또는 디렉토리 간 차이점을 찾고 보여주는 패키지입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Findutils-4.7.0

파일을 찾는 프로그램이 포함되어있습니다. 재귀적으로 검색하고 데이터베이스를 생성하고 유지 관리 및 검색을 할 때 사용됩니다. dpkg 설치를 진행할 때 필수입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Gawk-5.0.1

텍스트 파일을 조작하는 프로그램이 포함되어 있습니다. dpkg 설치를 위한 필수 패키지입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Grep-3.4

파일 검색을 위한 프로그램이 포함되어 있습니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Make-4.3

패키지 컴파일을 위한 패키지입니다. 필수입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

구성 옵션의 의미: --without-guile : make 패키지 내의 chroot 환경에서 사용이 불가능한 guile 라이브러리를 비활성화합니다.

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Perl-5.30.1

동적 타입의 스크립트 언어인 perl을 지원하는 패키지입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

컴파일 합니다.

```
make
```

그 중 다 설치하지 않고 몇 가지만 설치합니다.

```
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
```

Python-3.8.1

파이썬과 파이썬의 개발 환경이 포함되어 있습니다.

<* 파이썬에 대한 파일이 2가지가 있는데 그 중 앞글자가 대문자인 Python-3.8.1.tar.xz를 풀어 진입하면 됩니다. python-3.8.1-docs-html.tar.bz2 라는 파일은 이 단계와 무관하니 주의하여 진행하시면 됩니다.>**

이 패키지에서 먼저 python 인터프리터를 빌드한 다음 일부 표준 python 모듈을 빌드합니다.

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

컴파일을 위해 구성 옵션을 조정합니다.

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

구성 옵션의 의미: --without-ensurepip : pip 패키지 설치 프로그램은 이 단계에서 필요없기 때문에 설치하지 않습니다.

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Sed-4.8

스트림 편집기가 포함되어 있습니다. 향후 dpkg을 설치하는 데 필수입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Tar-1.32

tar 보관 프로그램에 대해 포함하고 있는 패키지입니다.

컴파일을 위해 구성 옵션을 조정합니다.

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

패키지를 컴파일합니다.

```
make
```

패키지를 설치합니다.

```
make install
```

Stripping

지금까지 temporary tools를 구성하기 위한 필수 패키지를 전부 설치하였습니다. 이 중 불필요한 파일을 제거하는 stripping을 진행하여 무게를 줄입니다. 최대 70MB까지 제거가 가능합니다. 밑 코드들을 차례로 진행합니다.

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

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

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

6. 필요한 패키지 설치(dpkg, apt 디펜던시)

챕터 6에서는 dpkg 소스코드 그리고 apt를 설치하기 위한 모든 디펜던시 패키지를 설치합니다.

모든 패키지는 만들고자 하는 최종 시스템의 아키텍쳐에 따라 다를 수 있으며

현재 문서는 amd64를 타겟으로 진행합니다.

dpkg 소스 파일 설치

source 디렉토리안에 debian 패키지 설치를 위한 디렉토리를 만듭니다.

`mkdir -v build $LFS/source/debian`

`cd $LFS/source/debian`

dpkg source를 다운로드 합니다.

`wget http://http.debian.net/debian/pool/main/d/dpkg/dpkg_1.17.27.tar.xz`

apt 디펜던시 파일 설치

apt의 디펜던시인 .deb파일들을 아래 링크에서 본인의 타겟 아키텍쳐에 맞게 설치합니다.

(남은 과정은 amd64를 타겟으로 진행되었습니다.)

[apt] https://packages.debian.org/jessie/apt

[debian-archive-keyring] https://packages.debian.org/jessie/debian-archive-keyring

[dpkg] https://packages.debian.org/jessie/dpkg

[gcc-4.9-base] https://packages.debian.org/jessie/gcc-4.9-base

[gnupg] https://packages.debian.org/jessie/gnupg

[gpgv] https://packages.debian.org/jessie/gpgv

[libacl1] https://packages.debian.org/jessie/libacl1

[libapt-pkg4.12] https://packages.debian.org/jessie/libapt-pkg4.12

[libattr1] https://packages.debian.org/jessie/libattr1

[libbz2-1.0] https://packages.debian.org/jessie/libbz2-1.0

[libc6] https://packages.debian.org/jessie/libc6

[libgcc1] https://packages.debian.org/jessie/libgcc1

[liblzma5] https://packages.debian.org/jessie/liblzma5

[libpcre3] https://packages.debian.org/jessie/libpcre3

[libreadline6] https://packages.debian.org/jessie/libreadline6

[libselinux1] https://packages.debian.org/jessie/libselinux1

[libstdc++6] https://packages.debian.org/jessie/libstdc++6

[libinfo5] https://packages.debian.org/jessie/libtinfo5

[libusb-0.1-4] https://packages.debian.org/jessie/libusb-0.1-4

[multiarch-support] https://packages.debian.org/jessie/multiarch-support

[readline-common] https://packages.debian.org/jessie/readline-common

[tar] https://packages.debian.org/jessie/tar

[zlib1g] https://packages.debian.org/jessie/zlib1g

소유권 변경

이제 더 이상 lfs에서 작업하지 않습니다. 여기서부터는 root 계정으로 진입하여 진행할 것입니다.

현재 lfs계정에서 root계정으로 나갑니다.

```
exit
```

사용자의 소유권을 이임합니다.

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

7. Chroot 환경 진입

챕터 7에서는 Chroot 환경에 진입하는 것을 목표로 합니다.

Chroot 환경에 진입하면 LFS 시스템에서만 모든 일을 진행하게 됩니다.

가상 커널 파일시스템 준비

```
mkdir -pv $LFS/{dev,proc,sys,run}
mknod -m 600 $LFS/dev/console c 5 1
mknod -m 666 $LFS/dev/null c 1 3
mount -v --bind /dev $LFS/dev
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

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

구성 명령의 의미: mkdir -pv $LFS/dev, proc, sys, run : 파일시스템이마운트될디렉토리를만듭니다. mknod − m600LFS/dev/console c 5 1; mknod -m 666
LFS/dev/null c13 : 커널이시스템을부팅할때커널은몇가지디바이스노드를필요로합니다. 디바이스노드는반드시하드디스크에생성되어야하기때문에. 디바이스노드들을udevd가시작되기전에사용가능하게끔합니다. mou
LFS/dev : 새로운 시스템은 Udev를 가지고 있지 않고 아직 부팅 되지 않는 상태이므로 /dev에 마운트를 수동으로 해주어야 합니다. mount -vt devpts devpts LFS/dev/pts − ogid = 5, mode = 620; mount − vtprocprocLFS/proc; mount -vt sysfs sysfs LFS/sys; mount − vttmpfstmpfsLFS/run : 이제 남은 가상 커널 파일 시스템을 마운트합니다. if [-h LFS/dev/shm]; thenmkdir − pv LFS/(readlinkLFS/dev/shm) fi : 몇 호스트 시스템에서는 /dev/shm이 /run/shm의 심볼릭 링크입니다. 이런 경우 /dev/shm 디렉토리를 만들어 줘야 하기 때문에 진행합니다.

Chroot 환경 들어가기

Chroot 환경에 진입합니다.

```
chroot "$LFS" /tools/bin/env -i \
HOME=/root                  \
TERM="$TERM"                \
PS1='\[\033[01m\][ \[\033[01;34m\]\u@\h\[\033[00m\]\[\033[01m\]]\[\033[01;32m\]\w\[\033[00m\]\n\
[\033[01;34m\]$\[\033[00m\]> ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin \
/tools/bin/bash --login +h
```

8. dpkg compile및 apt 설치

챕터 8에서는 dpkg와 apt를 설치합니다. dpkg는 모든 로컬에 있는 데비안 계열 패키지를 관리하고, apt는 외부 패키지를 설치하거나 업데이트할 수 있습니다.

apt설치를 모두 dpkg를 통해 진행하므로(.deb file) dpkg설치를 끝낸 후 apt 설치를 진행합니다.

dpkg compile

bash linking

temporary tools 의 bash를 기본 쉘로 지정하기 위해 bash를 링크합니다.

```
mkdir /bin
ln /tools/bin/bash /bin/bash
ln /tools/bin/bash /bin/sh
```

dpkg 설치하기

dpkg source file의 압축을 풀고 해당 디렉토리로 이동합니다.

그런 다음 환경설정을 한 뒤 컴파일과 설치를 진행합니다.

```
cd /source/debian/dpkg-1.17.27
./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var --build=x86_64-unknown-linux-gnu
make
make install
```

설치가 끝나면 dpkg디렉토리를 삭제합니다.

```
cd ..
rm -Rf dpkg-1.17.27
```

dpkg database 만들기

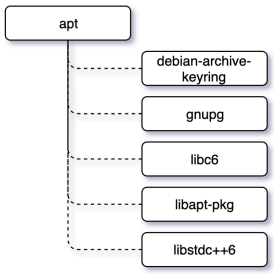
dpkg는 로컬환경에 설치된 모든 debian package들을 관리합니다. 그러기 위해서는 dpkg가 데이터베이스로 사용할 파일을 만들어 줘야 합니다.

```
touch /var/lib/dpkg/status
```

apt 설치

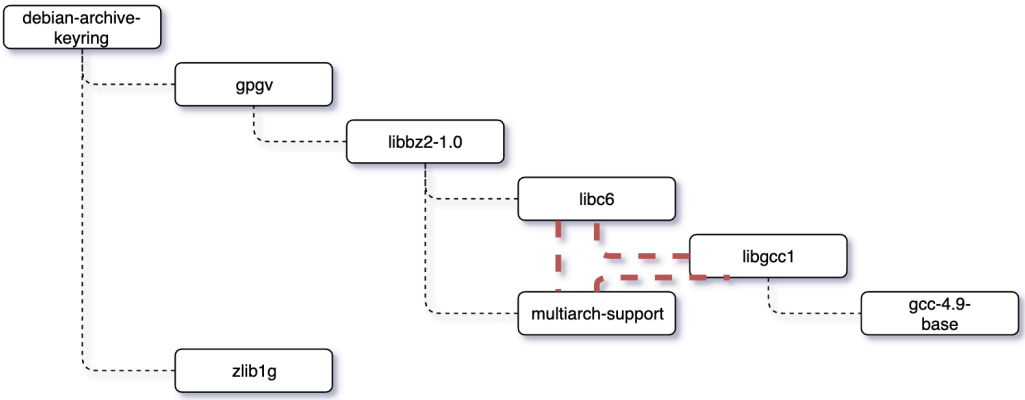
apt의 디펜던시는 다음과 같이 구성되어 있습니다.

apt설치에 앞서 5가지 디펜던시를 모두 설치해주어야 합니다.



debian-archive-keyring설치하기

첫번째 디펜던시인 debian-archive-keyring설치를 시작합니다. 하지만 여기서 문제가 발생합니다. libc6은 libgcc1에 의존하고, libgcc1은 multiarch-support에 의존합니다. 또한 multiarch-support는 libc6에 의존합니다. 결국 순환 구조의 디펜던시에 관련된 문제가 발생하는 것입니다.



위 같은 문제점이 존재함을 고려하며 설치를 진행하겠습니다.

이제 다음과 같은 명령어로 deb package들을 설치합니다.

```
dpkg -i (location of gcc-4.9-base)
```

libgcc1을 설치했을때 libgcc1의 디펜던시가 설치되어있지 않아. libgcc1의 일부만 설치합니다.

```
dpkg - i (LOCATION_OF_libgcc1)
```

sed 커맨드를 통해 libgcc1이 설치된 것처럼 시스템을 속입니다.

왜냐하면 그래야 libgcc1에 의존하고 있는 다른 패키지들을 설치할 수 있기 때문입니다.

```
sed -ir 's/not-installed/installed/' /var/lib/dpkg/status
```

multiarch package를 설치합니다.

```
dpkg -i (location_of_multiarch)
```

하지만 여기서 다음과 같은 에러가 발생할 수 있습니다.

```
$> dpkg -i multiarch-support_2.19-18+deb8u10_amd64.deb
dpkg: warning: parsing file '/var/lib/dpkg/status' near line 22 package 'libgcc
1':
 missing description
dpkg: warning: parsing file '/var/lib/dpkg/status' near line 22 package 'libgcc
1':
 missing maintainer
dpkg: error: parsing file '/var/lib/dpkg/status' near line 22 package 'libgcc1'
:
 missing version
```

이런 경우가 발생했을 때에는 먼저 logout으로 chroot 환경을 logout합니다.

```
logout
```

그다음 다음과 같은 명령어로 status 파일을 수정하기 위해 들어갑니다.

```
vi $LFS/var/lib/dpkg/status
```

<*주의사항 : chroot environment를 나가면 환경변수 설정이 풀리는 경우가 있습니다. echo $LFS로 환경변수 설정을 확인하고 설정이 풀린경우 `export LFS=/mnt/lfs` 를 입력합니다.>**

파일 내 libgcc1 패키지 부분에,

```
Package: libgcc1
Status: install ok installed
Priority: required
Section: libs
Architecture: amd64
```

```
Description: GCC support library

 Shared version of the support library, a library of internal subroutines

 that GCC uses to overcome shortcomings of particular machines, or

 special needs for some languages.

Maintainer: Debian GCC Maintainers <debian-gcc@lists.debian.org>

Version: 1:4.9.2-10+deb8u2
```

위 코드를 더합니다. 버전은 설치한 libgcc1에 따라 바뀔 수 있습니다.

```
Package: libgcc1
Status: install ok installed
Priority: required
Section: libs
Architecture: amd64
Description: GCC support library
 Shared version of the support library, a library of internal subroutines
 that GCC uses to overcome shortcomings of particular machines, or
 special needs for some languages.
Maintainer: Debian GCC Maintainers <debian-gcc@lists.debian.org>
Version: 1:4.9.2-10+deb8u2
```

해당 패키지에 부족한 내용을 채워넣으면 위 그림과 같은 결과창을 보실 수 있습니다.

위 설정을 마친 뒤 다시 chroot 환경으로 진입하기 위해서는 가상 커널 파일시스템을 다시 마운트 해준 뒤 들어가야 합니다. (앞서 chroot 환경으로 진입하기 위해 진행한 과정과 같습니다.)

가상 커널 파일시스템을 다시 마운트합니다.

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

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

chroot 환경에 재진입합니다.

```
chroot "$LFS" /tools/bin/env -i \
HOME=/root                  \
TERM="$TERM"                \
PS1='\[\033[01m\][ \[\033[01;34m\]\u@\h\[\033[00m\]\[\033[01m\]]\[\033[01;32m\]\w\[\033[00m\]\n\
[\033[01;34m\]$\[\033[00m\]> ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin \
/tools/bin/bash --login +h
```

이어서 multiarch를 계속 설치합니다.

```
dpkg -i (location_of_multiarch)
```

위 명령을 쳤을 때 libc6이 아직 설치가 완료되지 않았으므로 warning이 뜰 것입니다. libc6도 multiarch의 디펜던시이기 때문입니다. (부분만 설치가 진행됩니다.)

libc6에 대해 설치를 진행해주어야 multiarch가 설치되기 때문에 밑 명령을 진행합니다.

```
dpkg -i (location_of_libc6)
```

이제는 multiarch가 완벽하게 설치가 진행됩니다.

```
dpkg --configure -a
```

이제 libgcc1의 디펜던시가 모두 준비됐으므로 재설치를 진행합니다.

`dpkg - i (LOCATION_OF_libgcc1)`

나머지 패키지들은 아래 명령을 dpkg가 설치를 끝내기 전까지 반복하면서 모든 디펜던시 트리가 설치됩니다.

```
dpkg -i *
```

다 설치가 되면, 확인합니다.

```
echo $(($(dpkg -l | wc -l)-5))
```

만약 23을 리턴한다면 apt가 제대로 설치된 것입니다.

9. Base system 구축

챕터 9에서는 데비안 시스템에 필수적인 패키지들을 설치하고, 필수적인 환경설정 파일들을 만듭니다.

베이스 패키지들을 설치함으로써 여러가지 리눅스 유틸리티 커맨드(ls, ln, mount, etc..)를 사용할 수 있으며

초기 init 시스템에서도 데비안 시스템이 잘 작동하게 합니다.

네트워크 설정

/etc/resolv.conf

DNS resolution을 하기 위해서는 /etc/resolv.conf 파일이 필요합니다.

```
cat > /etc/resolv.conf << "EOF"
nameserver 8.8.8.8
nameserver 8.8.4.4
EOF
```

/etc/apt/sources.list

sources.list는 apt가 컨택하는 저장소들을 모아놓은 파일입니다.

```
cat > /etc/apt/sources.list << "EOF"
# Debian Jessie main repos
deb http://httpredir.debian.org/debian/ jessie main
deb-src http://httpredir.debian.org/debian/ jessie main

#Debian Jessie security repos
deb http://security.debian.org/ jessie/updates main
deb-src http://security.debian.org/ jessie/updates main

# non-free plugins
deb http://http.debian.net/debian/ jessie non-free contrib main
```

```
# jessie-updates, previously known as 'volatile'
deb http://httpredir.debian.org/debian/ jessie-updates main
deb-src http://httpredir.debian.org/debian/ jessie-updates main
EOF
```

/etc/hosts

/etc/hosts파일은 호스트의 아이피주소를 담고 있습니다. 이것은 사용자의 ipv4, ipv6 loopback addresses를 포함해야 합니다.

(잘 모르시겠다면 호스트 머신의 /etc/hosts파일을 참고하시거나 아래 예시와 같이 그대로 가셔도 됩니다.)

```
cat > /etc/hosts << "EOF"
127.0.0.1       localhost

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
EOF
```

/etc/hostname

/etc/hostname은 호스트의 DNS 이름을 포함하고 있습니다. 우리는 debianfromscratch로 하는 것으로 정하겠습니다.(원하시는 이름으로 변경 가능합니다.)

```
cat > /etc/hostname << "EOF"
debianfromscratch
EOF
```

apt의 패키지 리스트 업데이트

apt의 패키지 리스트를 업데이트합니다.

```
mkdir /tmp
apt-key update
apt-get update
```

유저 생성

debianutils

base-passwd 패키지를 설치를 위해 필요한 패키지를 설치합니다.

```
apt-get install debianutils
```

base-passwd

모든 데비안 시스템에서 사용되는 /etc/passwd, /etc/group을 제공하는 패키지를 설치합니다.

```
apt-get install base-passwd
```

/etc/gshadow , /etc/shadow 파일 만들기

passwd 패키지가 관리하는 파일을 수동으로 만들어 주어야 합니다.

```
touch /etc/shadow /etc/gshadow`
```

login

login 패키지를 설치하면 시스템에 새로운 세션을 만들 수 있는 기능을 우리에게 제공합니다. 이를 설치합니다.

```
apt-get install login`
```

passwd

passwd 패키지를 설치하면 유저와 그룹의 정보를 조작할 수 있습니다. 설치합니다.

```
apt-get install passwd
```

adduser

유저를 추가할 시 필요한 패키지를 설치합니다.

```
apt-get install adduser
```

이제 root사용자의 패스워드를 설정합니다.

```
passwd root
pwconv
```

/etc/inputrc 파일 만들기

/etc/inpurc는 libreadline6 라이브러리가 사용하는 글로벌 환경설정 파일입니다.

```
cat > /etc/inputrc << "EOF"
# /etc/inputrc - global inputrc for libreadline
# See readline(3readline) and `info rluserman' for more information.

# Be 8 bit clean.
set input-meta on
set output-meta on

# To allow the use of 8bit-characters like the german umlauts, uncomment
# the line below. However this makes the meta key not work as a meta key,
# which is annoying to those which don't need to type in 8-bit characters.

# set convert-meta off

# try to enable the application keypad when it is called.  Some systems
# need this to enable the arrow keys.
# set enable-keypad on

# see /usr/share/doc/bash/inputrc.arrows for other codes of arrow keys

# do not bell on tab-completion
# set bell-style none
# set bell-style visible

# some defaults / modifications for the emacs mode
$if mode=emacs

# allow the use of the Home/End keys
"\e[1~": beginning-of-line
"\e[4~": end-of-line

# allow the use of the Delete/Insert keys
"\e[3~": delete-char
"\e[2~": quoted-insert

# mappings for "page up" and "page down" to step to the beginning/end
# of the history
# "\e[5~": beginning-of-history
# "\e[6~": end-of-history

# alternate mappings for "page up" and "page down" to search the history
# "\e[5~": history-search-backward
# "\e[6~": history-search-forward

# mappings for Ctrl-left-arrow and Ctrl-right-arrow for word moving
"\e[1;5C": forward-word
"\e[1;5D": backward-word
"\e[5C": forward-word
"\e[5D": backward-word
"\e\e[C": forward-word
"\e\e[D": backward-word
```

```
$if term=rxvt
"\e[7~": beginning-of-line
"\e[8~": end-of-line
"\eOc": forward-word
"\eOd": backward-word
$endif

# for non RH/Debian xterm, can't hurt for RH/Debian xterm
# "\eOH": beginning-of-line
# "\eOF": end-of-line

# for freebsd console
# "\e[H": beginning-of-line
# "\e[F": end-of-line

$endif
EOF
```

베이스 패키지 설치

이제 베이스 패키지를 apt 으로 설치하는 과정입니다.

debian filesystem hierarchy 만들기

debian filesystem hierarchy를 만들기 위해 밑 명령을 진행합니다.

```
rm -rf /var/mail
apt-get install base-files
```

필수적인 패키지 설치

기본적인 리눅스 유틸리티 패키지와 초기 init 패키지, 커널 컴파일에 필요한 필수 패키지들을 설치합니다.

```
apt-get install coreutils bc init util-linux libc-bin bsdutils grep kmod busybox
```

Chromium 설치

Chromium 을 설치합니다.

```
apt-get install chromium
```

10. 커널 컴파일

챕터 10에서는 운영체제의 핵심인 커널을 컴파일하게 됩니다.

리눅스 커널은 버전이 다양합니다.

취향과 상황에 맞는 리눅스 커널 버전을 고를 수 있으며 커널 옵션 또한 커스터마이징이 가능합니다.

커널 소스코드 다운로드

원하는 커널 버전을 설치합니다. 이 때 저희는 아직 커널 소스코드를 다운받지 않은 상태이므로, logout을 통해 chroot을 나간 후 원하는 버전의 커널 소스코드를 다운 받아야 합니다.

```
logout
```

https://www.kernel.org/ <-이 사이트에서 원하는 버전의 커널 설치가 가능합니다.

<* chroot을 나가고 나서는 항상 $LFS 의 환경변수가 잘 설정되어 있는지 잘 확인해야 합니다!!! chroot 환경에 로그인 및 로그아웃 할 시 `echo=$LFS` 명령을 입력해 항상 확인합니다.>**

커널 소스코드를 다운받았다면, 아래 코드를 통해 chroot을 재진입합니다.

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

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

```
fi

chroot "$LFS" /tools/bin/env -i \
HOME=/root                  \
TERM="$TERM"                \
PS1='\[\033[01m\][ \[\033[01;34m\]\u@\h\[\033[00m\]\[\033[01m\]]\[\033[01;32m\]\w\[\033[00m\]\n\
[\033[01;34m\]$\[\033[00m\]> ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin:/tools/sbin \
/tools/bin/bash --login +h
```

커널 컴파일 및 설치

다운받은 커널 소스코드의 압축을 푼 다음 해당 커널 소스 디렉토리로 이동합니다.

```
make mrproper
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /usr/include
```

커널 설정은 일단 디폴트 설정을 하는 것을 추천합니다. 그 다음 menuconfig으로 커스터마이징이 가능합니다.

```
make defconfig
```

그 다음 커널 설정을 커스터마이징합니다. (필요한 경우에만 진행합니다.)

```
make menuconfig
```

<*만약 당신이 VMware에서 이 과정을 진행하고 있는 상태라면 반드시 다음과 같은 옵션을 체크해주어야 합니다 !!! >**

Device Driver - Misc Devices - Vmware VMCI driver[*]

Device Driver - Fusion MPT device support - Fusion MPT ScsiHost drivers for SPI (used by VMware Player and workstation by default) [*]

모든 준비가 끝났으면 커널 컴파일을 시작합니다. 다소 시간이 많이 걸릴 수 있습니다. (가지고 있는 코어의 개수만큼 -j{코어의 개수} 플래그를 make 뒤에 붙여서 컴파일을 빠르게 진행할 수 있습니다. (ex. make -j4))

```
make
```

패키지 설치가 끝났으면 커널 모듈을 설치합니다.

```
make modules_install
```

컴파일이 완료된 커널을 /boot directory로 복사합니다.

```
cp -v arch/x86/boot/bzImage /boot/vmlinuz-dfs
cp -v System.map /boot/System.map-4.4.2
cp -v .config /boot/config-dfs
```

11. 부팅 가능한 형태로 만들기

챕터 11에서는 부팅 가능한 LFS 형태로 제작합니다.

부팅에 필요한 grub설정과 파일 시스템이 잘 마운트 될 수 있도록 환경 설정 파일들을 제작합니다.

환경설정 및 재부팅

파일시스템이 정상적으로 마운트 되기 위해서는 /etc/fstab파일이 필요합니다.

아래 예시는 sdb1이 루트파일 시스템이고 ext4파일시스템으로 마운트 되어있습니다. 그리고 sdb2는 스왑파티션으로 나눠논 파티션이기 때문에 다음과 같이 마운트 되어있습니다.

이는 실습자의 환경에 따라 달라질 수 있음을 인지하고 주의해야 합니다. (현재 문서를 진행하는 파티션의 이름, 스왑 파티션의 여부 등등)

```
cat > /etc/fstab << "EOF"

\# Begin /etc/fstab
```

```
\# file system mount-point type   options       dump fsck

\#                               order


/dev/sdb1   /      ext4  defaults      1   1

/dev/sdb2   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
```

chroot에서 필요한 모든 과정은 끝났습니다. logout 커맨드를 통해 chroot밖으로 나갑니다.

```
logout
```

grub설정을 바꿔줌으로써 새로운 파티션으로 부팅이 될 수 있게끔 grub.cfg를 수정합니다.

(아래 설정은 새로운 파티션으로만 부팅이 가능하게끔 설정되어있으므로 그걸 원하지 않는 다면 현재 본인 Host machine의 /boot/grub/grub.cfg를 참고하여 아래내용을 기존의 grub.cfg 파일 위에 추가해주면 됩니다.)

(또한 아래 예시는 sdb1 (hd1,1) 이 루트파티션으로 설정되어있는 경우를 가정합니다.)

```
cat > /boot/grub/grub.cfg << "EOF"

\# Begin /boot/grub/grub.cfg

set default=0

set timeout=5



insmod ext2

set root=(hd1,1)



menuentry "Debian from Scratch GNU/Linux" {

    linux  /boot/vmlinuz-dfs root=/dev/sdb1 ro

}

EOF
```

마지막으로 모든 LFS에 마운트 되어있는 것들을 unmount합니다.

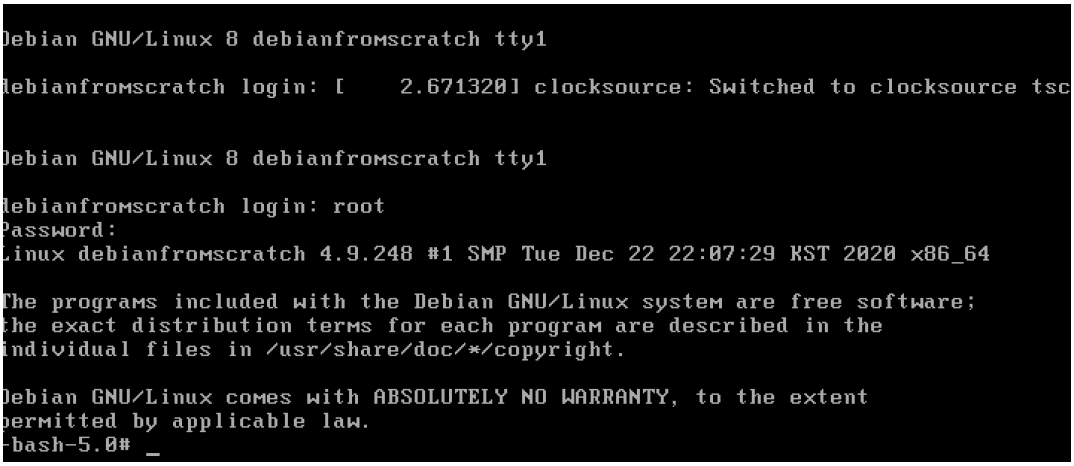
```
umount -v $LFS/dev/pts
umount -v $LFS/dev
umount -v $LFS/run
umount -v $LFS/proc
umount -v $LFS/sys
umount -v $LFS
```

(umount가 완벽히 되지않아도 크게 신경쓰시지 않으셔도 됩니다.)

마지막으로 시스템을 재부팅합니다.

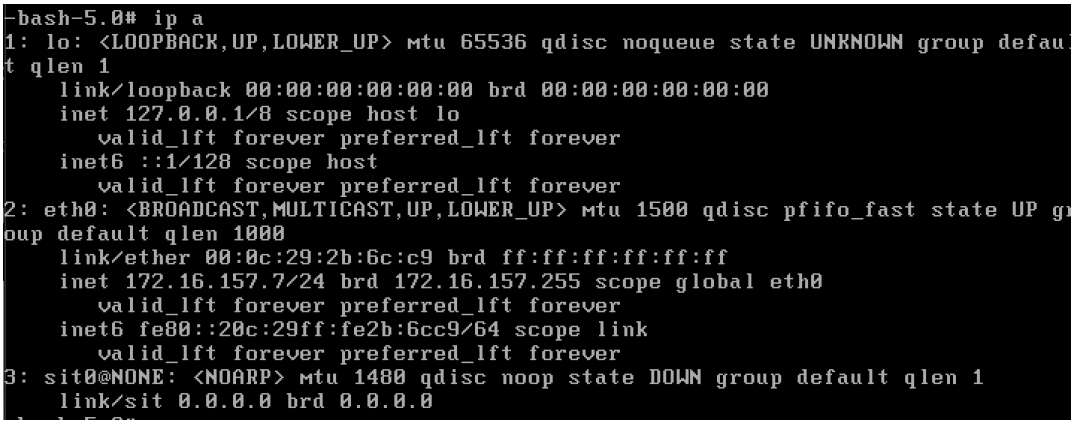
```
shutdown -r now
```

재부팅이 돼서 다음과 같은 화면이 나오면 정상입니다. (현재 사용자가 root밖에 없으므로 root로 로그인합니다.)



마지막으로 네트워크 설정은 아래 명령을 입력해 본인의 디바이스가 가지고 있는 네트워크 인터페이스 카드를 확인합니다.

```
ip a
```



위 같은 경우는 eth0인 것을 확인할 수 있습니다.

```
dhclient eth0
```

다음과같이 eth0에 ip를 동적할당 해주게 되면 정상적으로 네트워크가 작동하는 것을 확인할 수 있습니다.

끝.