手把手教你如何建立自己的Linux系统 第二版

作者：孙海勇

更新日志（具体更新内容见文末）

2008年7月7日：增加内核支持显示UTF-8编码文字

2008年7月6日：修改笔误两处

2008年3月16日：修改笔误一处

2008年3月8日：修改命令一处

2008年3月3日：修改命令一处

2008年2月12日：修改/etc/profile的内容

2008年2月12日：修改笔误一处

2008年2月10日：本文发布。

前言：

这是本文的第二版本，据第一版本发布已经将近两年的时间，第一版针对LFS-6.1.1的手册进行讲解，LFS-6.3相对于LFS-6.1.1方法上有了一些变化，软件包也大量更新，为了方便新手能够快速进入状态，决定再次撰写此文的第二版，并针对LFS-6.3来编写。

本文在内容和形式上完全继承第一版本的风格，内容上根据版本进行变化，但整体没有本质的变化，在这里首先感谢哪些在第一版中提出意见和问题的网友，使得本文变的越来越完善。

LFS是一部非常好的制作一个完整的操作系统的手册，但LFS是属于指导性的手册，因此它默认的前提条件是具备一定的Linux使用经验的用户群，所以它在每个软件包的安装部分只给出了在目录中的全部操作指令，而对于解压缩之类的则交给用户自己去解决，但对于很多第一次使用LFS的用户往往会出现一些不清楚某条命令应该是在哪里执行的问题，本文力图从实例上来解释这些问题。

本文虽然是采用VMWare Workstation 5.5的环境下制作的，但仍然符合使用真实机器上的过程，只是会在某些地方要根据具体机器进行更改，文中会在这些地方做出说明。

使用WMWare来写这篇文章是为了说明方便，因为VMWare在各种不同机器环境下模拟的虚拟设备几乎相同，所以用它来说明一些需要实际例子才说的清楚的地方非常合适，而且用VMWare来做即使出错也不会对真实的系统造成破坏，很适合新手使用，唯一的缺点就是速度慢了些，一般只有真实机器的一半左右的速度。

本文力争完成一个完整的制作命令，可以根据本文提供的命令顺序输入就可以完成LFS了，通常命令表示为

代码:

```
命令
```

由于制作过程比较漫长，特别是在一些比较慢的机器上，关于在制作过程中重新启动后恢复到工作状态的方法在文中有详细的介绍，但由于篇幅比较长可能看起来比较麻烦，可以看我专门将这部分提取出来后完成的一篇**《制作LFS过程中各个阶段恢复工作状态的方法 第二版(适合LFS6.3)》**，内容比较集中容易查看。

恢复工作的方法更加适合在真实机器上制作LFS的朋友，如果使用VMWare也可以直接使用VMWare的暂停功能来保存现场，继续的时候恢复现场就可以了。

准备工作：

下载LiveCD的ISO文件(因为在制作本文时最新正式版只有6.3-r2145下载)：http://ftp.osuosl.org/pub/lfs-livecd/lfslivecd-x86-6.3-r2145.iso

刻录ISO文件到光盘上，如果你是用真实机器当然少不了这步，不过如果你用VMWare的话，就可以直接使用ISO文件了。

以VMWare Workstation 5.5为例（真实机器可跳过此部分）

选择File->New->Virtual Machine...启动向导

选择Custom，并选择New-Workstation 5然后在选择Guest operating system里选择Linux，在Version里选择Other Linux 2.6.x kernel

存放目录、处理器数量和内存大小根据实际情况，建议内存不得小于128M，最好256M以上

相关知识点：

LFS-6.3采用了GCC4.1.2，如果使用128M编译GCC4.x.x话就需要使用swap了，但如果配置了256M就可以在没有swap的情况下完成编译，所以条件允许的情况下使用256M。

Network connection里选择Use network address translation(NAT)

SCSI Adpters选择默认的LSI Logic就可以了

注意点：

这里选择的磁盘类型对于最后编译内核使用选项是有影响的，我在本文的第一版中使用BusLogic来建立系统，因此这里选择使用LSI Logic来用，如果使用BusLogic的话可以参考本文第一版中的内核编译选项来代替后面的内核选项部分。

选择Create a new virtual disk

在Virtual Disk Type这步比较重要，你可以选择IDE也可以选择SCSI，但这里的选择直接影响到最后编译内核时的选项。这里以选择IDE为例子。

相关知识点：

如果选择了IDE，则内核的默认设置就可以支持，但如果选择了SCSI，就必须在内核中加入对SCSI Adpters的支持，因为前面选择了LSI Logic，所以内核中就必须加入对LSI Logic的支持，否则将无法启动，相关部分在最后的内核编译部分有说明。

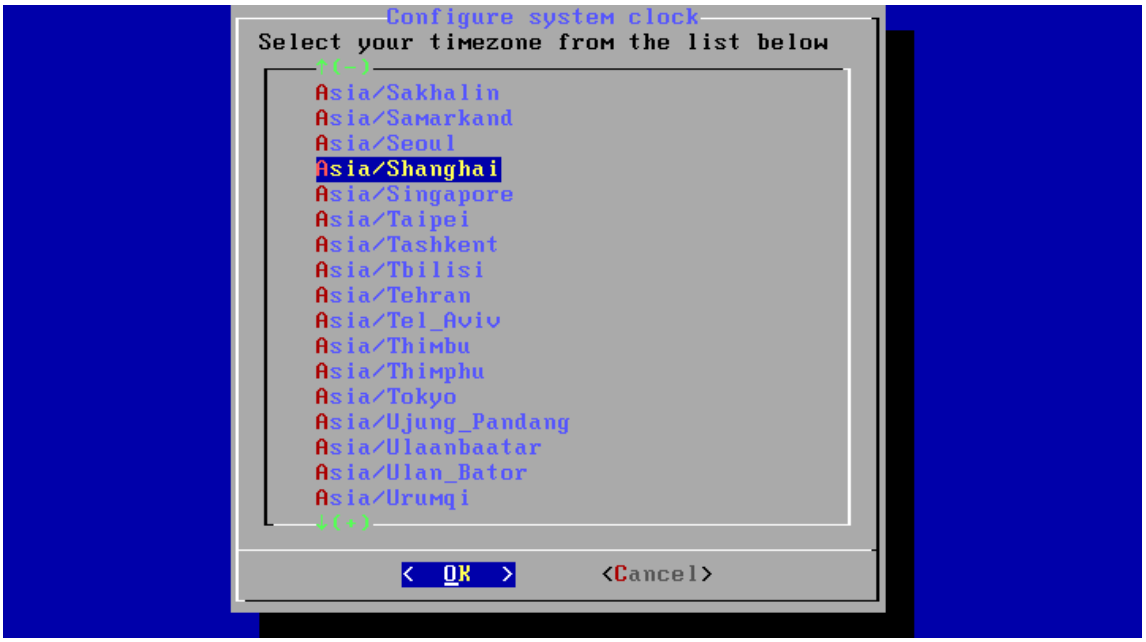
不过这里我建议选择使用IDE，一方面方便驱动，另外似乎LFS-6.3-r2145尚不能支持LSI Logic的SCSI磁盘，用该LiveCD启动后可能无法识别出硬盘来。

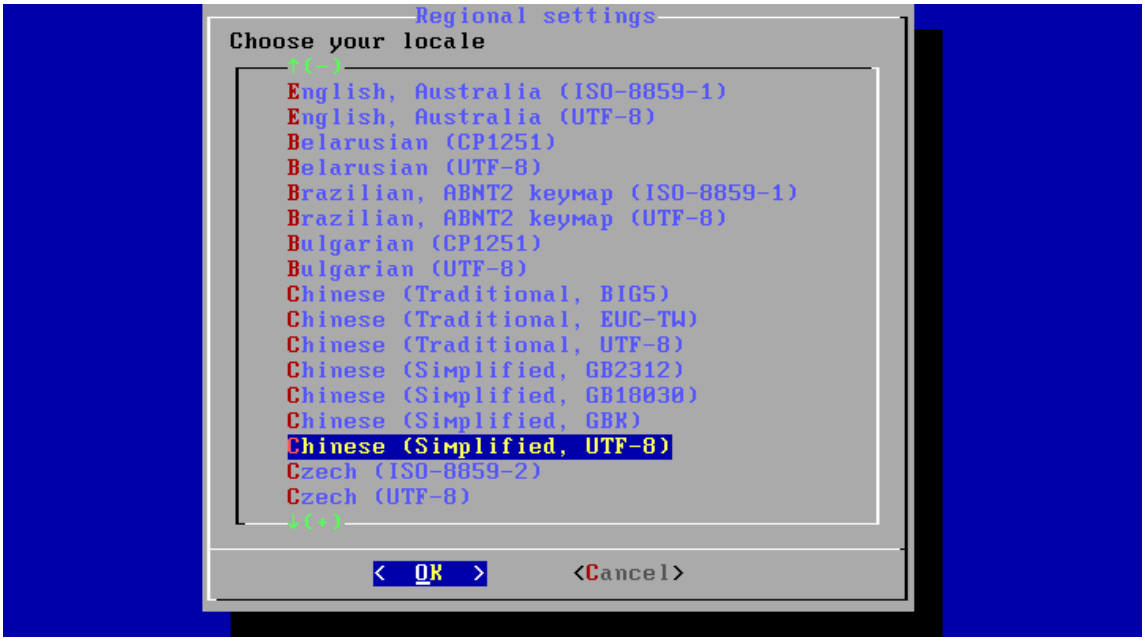
对于磁盘大小，使用4G足够编译LFS了，但如果你打算编译更多的BLFS，这里可以考虑适当的增加一些大小，如果磁盘空间比较富裕就用默认的8G好了。

完成向导后在虚拟机的界面里选择Edit virtual machine settings，将CD-ROM改为Use ISO image，然后选择LiveCD的ISO文件，如果你已经刻录好了光盘，将光盘放入光驱就行了。

点Start this virtual machine开是虚拟机

由于虚拟盘上没有任何信息，因此将自动从LiveCD中启动，在启动过程中会出现选择时区等信息，你可以按照实际情况选择，也可以按照默认选择，简单点就是等待一会系统会自动进行选择。这里我选择时区为Asia/Shanghai，选择本地语言为Chinese (Simplified, UTF-8)，其它的都按默认选择了。





启动完成LiveCD后就开始建造自己的LFS的历程了。

这里先介绍以下两个LiveCD下的重要目录

/usr/share/LFS-BOOK-6.3-HTML目录存放的就是LFS手册了

/lfs-sources里面存放的就是建造LFS所需要的源码包，不需要到处下软件了。

磁盘分区：

输入命令：

代码:

```
cfdisk /dev/hda
```

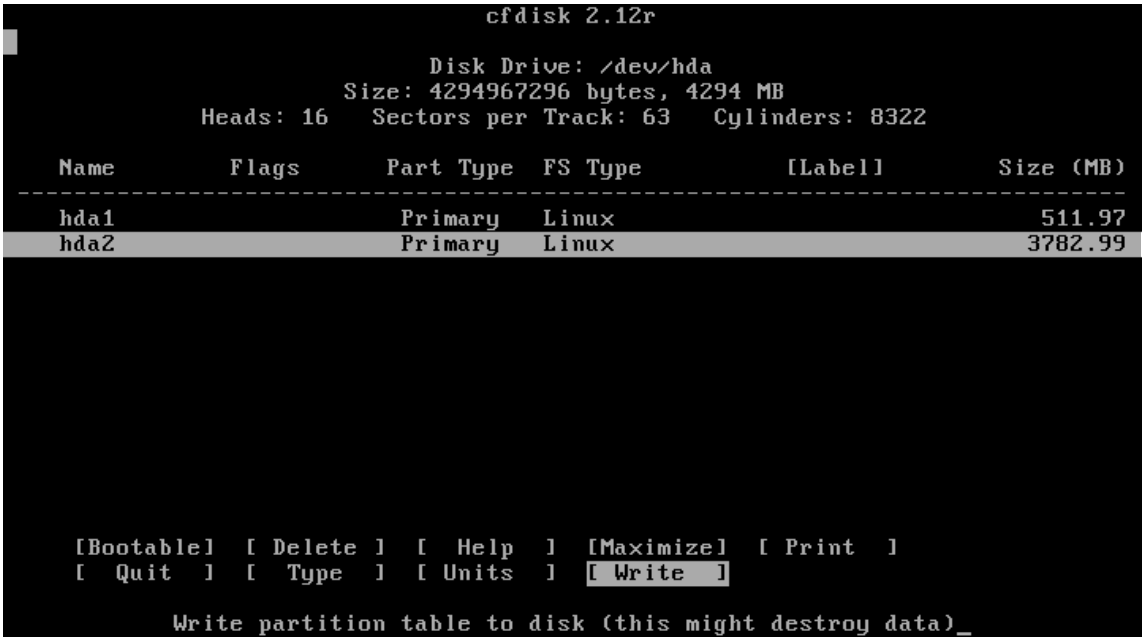
将出现分区界面

这里可以按照你自己的需要的分区，这里我按照设置一个根分区和一个交换分区为例，交换分区占用512M，其余的全部分给根分区。

磁盘分区 作用

/dev/hda1 swap

/dev/hda2 作为目标系统根目录



保存退出后进行磁盘分区的格式化

代码:

```
mkswap /dev/hda1
mkfs.xfs /dev/hda2
```

相关知识点：

磁盘格式化一定要在磁盘分区未进行加载前进行。

mkswap是用于将磁盘分区格式化为交换分区的命令。

这里我将/dev/hda2格式化了为Xfs格式，如果你喜欢其它格式的文件系统，你可以使用相应的命令来格式化。

注意：这里要根据实际情况建立和设置分区，如果你不太清楚这个问题，请先不要开始，否则可能造成难以恢复的损失！（这里只是根据VMWare里面的情况做的例子，在VMWare中相对安全些，建议初学者在虚拟机中开始。）

如果你的内存不太大，想在编译期间就使用上交换分区的话，可使用下面的命令激活交换分区

swapon /dev/hda1

相关知识点：

swapon用于激活交换分区

swapoff用于将激活的交换分区停用

可以通过free命令来查看当前的内存使用情况

创建LFS的“创作基地”

代码:

```
export LFS=/mnt/lfs
mkdir -pv $LFS
```

相关知识点：

export LFS=/mnt/lfs这条命令的作用是为了后面引用“创作基地”的绝对路径方便而设置LFS这样的环境变量。

加载/dev/hda2到“创作基地”

代码:

```
mount /dev/hda2 $LFS
```

创建必要的目录并设置属性

创建源代码编译用目录

代码:

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

相关知识点：

chmod a+wt是将目录或文件的属性设置为1777，这样任何人都可以对其进行读写。

创建工具链目录

代码:

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

注意：

ln -sv $LFS/tools执行后应该会输出

`/tools' -> `/mnt/lfs/tools'

表示正确。

相关知识点：

上面这两句就建立了神奇的工具链目录（是工具链目录不是工具链），这样的创建方式是为了在创建工具链和使用工具链创建目标系统的时候对于工具链的位置都是/tools，这样可保证工具链的正常使用

创建lfs用户

代码:

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

设置lfs密码，设置为空就行了，也就是输入密码的时候直接回车就成。

代码:

```
passwd lfs
```

将tools和sources目录的用户改为lfs，以便后面使用lfs来操作这两个目录

代码:

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

登陆到lfs用户

代码:

```
su - lfs
```

这时候你会发现命令行提示符已经由#改为了$

相关知识点：

其实如果不使用lfs用root也是能完成工具链的，不过需要对root的环境变量进行修改，还要防止因为输入错误而导致覆盖主系统下的文件，所以LFS手册中制作工具链部分就是为了解决这种意外的发生而用lfs用户来建立工具链

建立lfs用户的环境

代码:

```
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
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL PATH
```

```
EOF
source ~/.bash_profile
```

这里利用了bash的环境变量的设置文件将lfs的环境设置为符合编译工具链要求的最少的环境参数

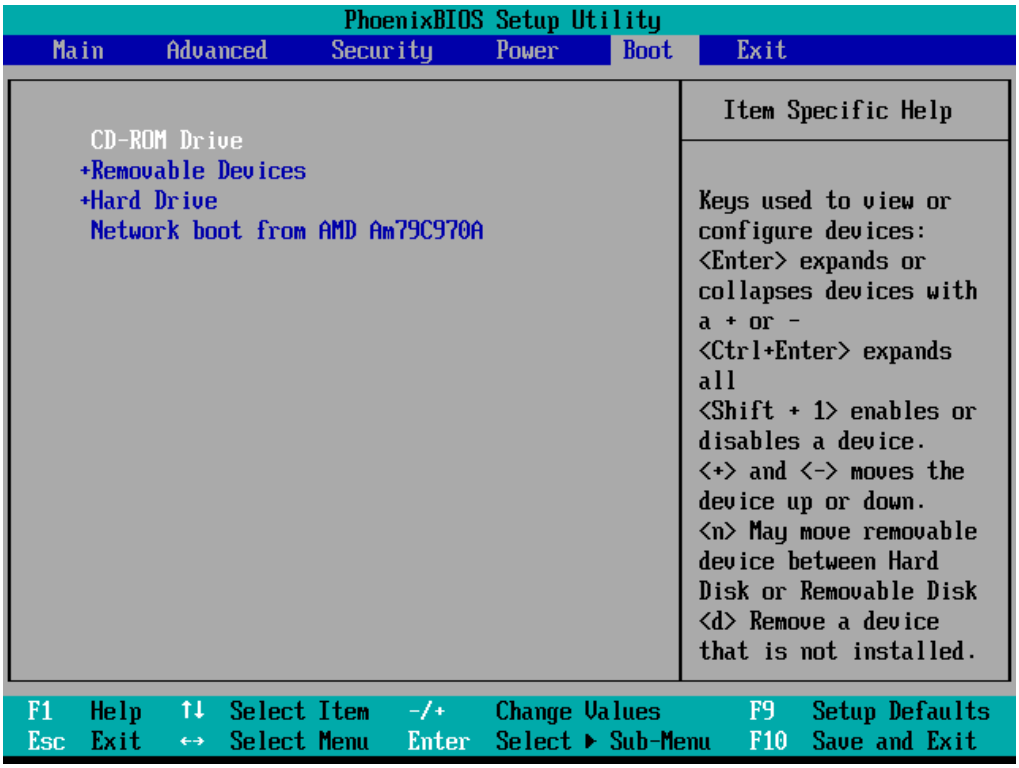
这里面最重要的就是PATH这个参数，目的是为了能够利用工具链里面的工具制作工具链：首先查找/tools/bin下是否有需要的命令，如果没有再到/bin和/usr/bin下找，然后用/bin或/usr/bin下面的命令来帮助生成需要的命令并放在/tools/bin下，这样此消彼涨，最终可完成一个自给自足的工具链。

到此为止就可以开始工具链的制作了，不过制作LFS是一个漫长而浩大的工程，所以要一直开机直到完成有时候比较困难，特别是在机器速度比较慢的情况下，能够重新启动到最后工作的状态是很重要的。在不同的阶段重新启动并恢复状态的步骤不完全相同，所以本文会在不同的阶段讨论重新启动恢复到工作状态的方法和步骤。

从现在开始一直到第五章结束，也就是完成Stripping中间的步骤中如果重新启动的恢复步骤：

1.重新启动计算机，并从LiveCD启动

相关知识点：在VMWare中因为磁盘已经有了信息了，所以会从磁盘启动，需要在启动虚拟机中的机器时按F2进入虚拟机的虚拟BIOS，然后在BOOT中设置第一启动为CD-ROM，保存退出即可。



2.LiveCD启动过程同第一次启动选择一样。

3.加载分区

```
export LFS=/mnt/lfs
mkdir -pv $LFS
mount /dev/hda2 $LFS
```

4.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

```
swapon /dev/hda1
```

5.建立工具链的链接

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

6.创建lfs用户

```
groupadd lfs
useradd -s /bin/bash -g lfs -m -k /dev/null lfs
passwd lfs
chown -v lfs $LFS/tools
chown -v lfs $LFS/sources
su - lfs
```

7.建立lfs用户的环境

```
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
PATH=/tools/bin:/bin:/usr/bin
export LFS LC_ALL PATH
EOF
source ~/.bash_profile
8.检查一下
export命令查看输出，应该是
declare -x HOME="/home/lfs"
declare -x LC_ALL="POSIX"
declare -x LFS="/mnt/lfs"
declare -x OLDPWD
declare -x PATH="/tools/bin:/bin:/usr/bin"
declare -x PS1="file:///u://w///$ "
declare -x PWD="/home/lfs"
declare -x SHLVL="1"
declare -x TERM="linux"
9.进入编译目录
cd $LFS/sources
基本上就恢复工作状态了。

开始工具链的制作

进入LFS包编译目录

代码:

```
cd $LFS/sources
```

Binutils-2.17 - Pass 1

代码:

```
tar xvf /lfs-sources/binutils-2.17.tar.bz2
cd binutils-2.17
```

相关知识点：

大家可以注意到后面所有的解包命令均使用tar xvf来完成，而不管文件的压缩方式是bz2还是gz，这是因为较新的tar程序都具有自动识别后缀名并自动调用相应的解压缩工具的能力，所以可以不需要指定压缩方式，但对于早期的tar命令则可能不具备这个功能因此需要你根据包的压缩方式来指定，如bz2使用j，gz使用z，对应上面的binutils则是tar xvjf /lfs-sources/binutils-2.17.tar.bz2

因LFS的LiveCD中提供的tar版本比较新，后面制作的tar版本也比较新，因此支持自动识别的能力，同时为了使文章的解压命令看起来比较统一方便维护（同样对于想制作成脚本的朋友也会比较方便）因此后面统一使用tar xvf来解压。

接着我们需要建立一个目录，因为binutils建议使用一个空目录来编译，所以

代码:

```
mkdir -v ../binutils-build
cd ../binutils-build
CC="gcc -B/usr/bin/" ../binutils-2.17/configure --prefix=/tools --disable-nls --disable-werror
make
make install
make -C ld clean
make -C ld LIB_PATH=/tools/lib
cp -v ld/ld-new /tools/bin
cd ..
rm -rf binutils-build
rm -rf binutils-2.17
```

GCC-4.1.2 - Pass 1

代码:

```
tar xvf /lfs-sources/gcc-4.1.2.tar.bz2
mkdir -v gcc-build
cd gcc-build
CC="gcc -B/usr/bin/" ../gcc-4.1.2/configure --prefix=/tools \
    --with-local-prefix=/tools --disable-nls \
    --enable-shared --enable-languages=c
make bootstrap
make install
ln -vs gcc /tools/bin/cc
cd ..
rm -rf gcc-build
rm -rf gcc-4.1.2
```

注意：这里不要图省事而不删**gcc-4.1.2**，因为这样可能会给后面的编译产生一些意外的错误。

Linux-2.6.22.5 API Headers

代码:

```
tar xvf /lfs-sources/linux-2.6.22.5.tar.bz2
cd linux-2.6.22.5
make mrproper
make headers_check
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /tools/include
cd ..
rm -rf linux-2.6.22.5
```

Glibc-2.5.1

代码:

```
tar xvf /lfs-sources/glibc-2.5.1.tar.bz2
cd glibc-2.5.1
mkdir -v ../glibc-build
cd ../glibc-build
../glibc-2.5.1/configure --prefix=/tools \
--disable-profile --enable-add-ons \
--enable-kernel=2.6.0 --with-binutils=/tools/bin \
--without-gd --with-headers=/tools/include \
--without-selinux
make
mkdir -v /tools/etc
touch /tools/etc/ld.so.conf
make install
cd ..
rm -rf glibc-build
rm -rf glibc-2.5.1
```

相关知识点：

这里的参数--enable-kernel=2.6.0，只是为了说明kernel的大版本，所以不需要根据实际的kernel版本来改，即使是用linux-2.6.15也一样只写2.6.0就可以了。

调整工具链

代码:

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

```
gcc -dumpspecs | sed 's@^/lib/ld-linux.so.2@/tools&@g' > `dirname $(gcc -print-libgcc-file-name)`/specs
GCC_INCLUDEDIR=`dirname $(gcc -print-libgcc-file-name)`/include &&
find ${GCC_INCLUDEDIR}/* -maxdepth 0 -xtype d -exec rm -rvf '{}' \; &&
rm -vf `grep -l "DO NOT EDIT THIS FILE" ${GCC_INCLUDEDIR}/*` &&
unset GCC_INCLUDEDIR
```

相关知识点：

工具链的调整方法有好几种，而且不同版本GCC的specs可能会有不同，但实际上都是把specs文件中的/lib/ld-linux.so.2替换成了/tools/lib/ld-linux.so.2，所以即使有些文章在调整工具链上的命令和LFS手册上的不一样也不用太奇怪，当然也可以直接用gcc -dumpspecs导出后手工直接编辑specs文件。

测试工具链的调整

echo 'main(){}' > dummy.c

cc dummy.c

readelf -l a.out | grep 'tools'

如果输出大致如下

[Requesting program interpreter: /tools/lib/ld-linux.so.2]

则表示调整成功，因为所有的库已经连接到了/tools/lib下。

rm -rf a.out dummy.c

测试工具安装

说明：这部分将安装3个用于第六章各种源码包编译后的测试的工具，所以如果你不打算做make check之类的事情，那么这3个包可以不装。

Tcl-8.4.15 Expect-5.43.0 DejaGNU-1.4.4

代码:

```
tar xvf /lfs-sources/tcl8.4.15-src.tar.gz
cd tcl8.4.15/unix
./configure --prefix=/tools
make
make install
make install-private-headers
ln -sv tclsh8.4 /tools/bin/tclsh
cd $LFS/sources
tar xvf /lfs-sources/expect-5.43.0.tar.gz
cd expect-5.43
patch -Np1 -i /lfs-sources/expect-5.43.0-spawn-1.patch
cp configure{,.bak}
sed 's:/usr/local/bin:/bin:' configure.bak > configure
./configure --prefix=/tools --with-tcl=/tools/lib --with-tclinclude=/tools/include --with-x=no
make
make SCRIPTS="" install
cd $LFS/sources
tar xvf /lfs-sources/dejagnu-1.4.4.tar.gz
cd dejagnu-1.4.4
./configure --prefix=/tools
make install
cd ..
rm -rf tcl8.4.15
rm -rf expect-5.43
rm -rf dejagnu-1.4.4
```

GCC-4.1.2 - Pass 2

代码:

```
tar xvf /lfs-sources/gcc-4.1.2.tar.bz2
cd gcc-4.1.2
cp -v gcc/Makefile.in{,.orig}
sed 's@\./fixinc\.sh@-c true@' gcc/Makefile.in.orig > gcc/Makefile.in
```

```
cp -v gcc/Makefile.in{,.tmp}
sed 's/^XCFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in.tmp \
> gcc/Makefile.in
patch -Np1 -i /lfs-sources/gcc-4.1.2-specs-1.patch
mkdir -v ../gcc-build
cd ../gcc-build
../gcc-4.1.2/configure --prefix=/tools \
--with-local-prefix=/tools \
--enable-clocale=gnu --enable-shared \
--enable-threads=posix --enable-__cxa_atexit \
--enable-languages=c,c++ --disable-libstdcxx-pch
make
make install
cd ..
rm -rf gcc-build
rm -rf gcc-4.1.2
```

再次测试工具链的调整，以确保刚刚编译的gcc正确工作

echo 'main(){}' > dummy.c

cc dummy.c

readelf -l a.out | grep 'tools'

如果输出大致如下

[Requesting program interpreter: /tools/lib/ld-linux.so.2]

则表示调整成功，因为所有的库已经连接到了/tools/lib下。

rm -rf a.out dummy.c

Binutils-2.17 - Pass 2

代码:

```
tar xvf /lfs-sources/binutils-2.17.tar.bz2
mkdir -v binutils-build
cd binutils-build
../binutils-2.17/configure --prefix=/tools --disable-nls \
 --with-lib-path=/tools/lib
make
make install
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
cp -v ld/ld-new /tools/bin
cd ..
rm -rf binutils-build
rm -rf binutils-2.17
```

Ncurses-5.6

代码:

```
tar xvf /lfs-sources/ncurses-5.6.tar.gz
cd ncurses-5.6
./configure --prefix=/tools --with-shared --without-debug --without-ada --enable-overwrite
make
make install
cd ..
rm -rf ncurses-5.6
```

Bash-3.2

代码:

```
tar xvf /lfs-sources/bash-3.2.tar.gz
cd bash-3.2
patch -Np1 -i /lfs-sources/bash-3.2-fixes-5.patch
./configure --prefix=/tools --without-bash-malloc
```

```
make
make install
ln -vs bash /tools/bin/sh
cd ..
rm -rf bash-3.2
```

Bzip2-1.0.4

代码:

```
tar xvf /lfs-sources/bzip2-1.0.4.tar.gz
cd bzip2-1.0.4
make
make PREFIX=/tools install
cd ..
rm -rf bzip2-1.0.4
```

Coreutils-6.9

代码:

```
tar xvf /lfs-sources/coreutils-6.9.tar.bz2
cd coreutils-6.9
./configure --prefix=/tools
make
make install
cp -v src/su /tools/bin/su-tools
cd ..
rm -rf coreutils-6.9
```

Diffutils-2.8.1

代码:

```
tar xvf /lfs-sources/diffutils-2.8.1.tar.gz
cd diffutils-2.8.1
./configure --prefix=/tools
make
make install
cd ..
rm -rf diffutils-2.8.1
```

Findutils-4.2.31

代码:

```
tar xvf /lfs-sources/findutils-4.2.31.tar.gz
cd findutils-4.2.31
./configure --prefix=/tools
make
make install
cd ..
rm -rf findutils-4.2.31
```

Gawk-3.1.5

代码:

```
tar xvf /lfs-sources/gawk-3.1.5.tar.bz2
cd gawk-3.1.5
./configure --prefix=/tools
cat >> config.h << "EOF"
#define HAVE_LANGINFO_CODESET 1
#define HAVE_LC_MESSAGES 1
EOF
make
```

```
make install
cd ..
rm -rf gawk-3.1.5
```

Gettext-0.16.1

代码:

```
tar xvf /lfs-sources/gettext-0.16.1.tar.gz
cd gettext-0.16.1
cd gettext-tools
./configure --prefix=/tools --disable-shared
make -C gnulib-lib
make -C src msgfmt
cp -v src/msgfmt /tools/bin
cd $LFS/sources
rm -rf gettext-0.16.1
```

Grep-2.5.1a

代码:

```
tar xvf /lfs-sources/grep-2.5.1a.tar.bz2
cd grep-2.5.1a
./configure --prefix=/tools --disable-perl-regexp
make
make install
cd ..
rm -rf grep-2.5.1a
```

Gzip-1.3.12

代码:

```
tar xvf /lfs-sources/gzip-1.3.12.tar.gz
cd gzip-1.3.12
./configure --prefix=/tools
make
make install
cd ..
rm -rf gzip-1.3.12
```

Make-3.81

代码:

```
tar xvf /lfs-sources/make-3.81.tar.bz2
cd make-3.81
./configure --prefix=/tools
make
make install
cd ..
rm -rf make-3.81
```

Patch-2.5.4

代码:

```
tar xvf /lfs-sources/patch-2.5.4.tar.gz
cd patch-2.5.4
./configure --prefix=/tools
make
make install
cd ..
rm -rf patch-2.5.4
```

Perl-5.8.8

代码:

```
tar xvf /lfs-sources/perl-5.8.8.tar.bz2
cd perl-5.8.8
patch -Np1 -i /lfs-sources/perl-5.8.8-libc-2.patch
./configure.gnu --prefix=/tools -Dstatic_ext='Data/Dumper Fcntl IO POSIX'
make perl utilities
cp -v perl pod/pod2man /tools/bin
mkdir -pv /tools/lib/perl5/5.8.8
cp -Rv lib/* /tools/lib/perl5/5.8.8
cd ..
rm -rf perl-5.8.8
```

Sed-4.1.5

代码:

```
tar xvf /lfs-sources/sed-4.1.5.tar.gz
cd sed-4.1.5
./configure --prefix=/tools
make
make install
cd ..
rm -rf sed-4.1.5
```

Tar-1.18

代码:

```
tar xvf /lfs-sources/tar-1.18.tar.bz2
cd tar-1.18
./configure --prefix=/tools
make
make install
cd ..
rm -rf tar-1.18
```

Texinfo-4.9

代码:

```
tar xvf /lfs-sources/texinfo-4.9.tar.bz2
cd texinfo-4.9
./configure --prefix=/tools
make
make install
cd ..
rm -rf texinfo-4.9
```

Util-linux-2.12r

代码:

```
tar xvf /lfs-sources/util-linux-2.12r.tar.bz2
cd util-linux-2.12r
sed -i 's@/usr/include@/tools/include@g' configure
./configure
make -C lib
make -C mount mount umount
make -C text-utils more
cp -v mount/{,u}mount text-utils/more /tools/bin
cd ..
rm -rf util-linux-2.12r
```

Stripping

这步是可有可无的，如果你打算今后还要用/tools里面的东西，那么可以strip一下来减少占用的磁盘空间，但如果做完目标系统后就删除了，不Strip也可以，反正最后也是要删掉的。

代码:

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

info和man里面的内容在制作过程中没什么用处，所以删掉也没啥关系。

代码:

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

退出lfs用户（这步不要少了）

代码:

```
exit
```

到目前为止，工具链已经制作完成了，接着就要开始制作真正的目标系统了，如果你到目前为止没出什么问题，那么恭喜你成功的通过了一关，不过接着还有相当长的路。

现在你应该是处于root用户状态的，看看你的命令行提示符是不是回到了#。

从现在开始不在需要lfs用户来制作系统了，因此我们用

代码:

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

重新设置目录权限，便于后面的工作。

创建三个重要目录

代码:

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

创建两个目标系统所必须的设备文件

代码:

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

这个时候也许你想睡觉关机了，那么重新开机后回到工作状态的步骤是：

1.重新启动计算机，并从LiveCD启动

2.加载分区

export LFS=/mnt/lfs

mkdir -pv $LFS

mount /dev/hda2 $LFS

3.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

swapon /dev/hda1

相关知识点：

这时候已经制作好了工具链，因此可以不需要建立根目录下的tools链接了。

利用主系统加载几个重要的文件系统，请注意这个步骤对于后面的工作极其重要。

代码:

```
mount -v --bind /dev $LFS/dev
mount -vt devpts devpts $LFS/dev/pts
mount -vt tmpfs shm $LFS/dev/shm
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
```

相关知识点：

mount命令加载的分区在重新启动后就失效了，所以在这其中重新启动则需要重新加载。

这里为了方便使用源码包，我将光盘加载到目标系统里

mkdir $LFS/cdrom

mount /dev/cdrom $LFS/cdrom

这个步骤不是必须的，如果你想使用，那么在重新启动后进入工作状态的步骤中在相应的位置上加入。

这里有一个更简单的办法，将lfs-sources里面所有源码包复制到$LFS/sources目录中，这步对于后面在第六章使用源代码将非常方便。

代码:

```
cp -a /lfs-sources/* $LFS/sources/
```

关于增加中文显示功能：

为了方便在制作完后的系统能够直接显示中文，这里可以从网络上下载本人写的一个显示UTF-8编码文字的内核补丁。

使用下面的命令来下载：

cd $LFS/sources/

wget http://zdbr.net.cn/download/utf8-kernel-2.6.22.5-core-1.patch.bz2

wget http://zdbr.net.cn/download/utf8-kernel-2.6.22.5-fonts-1.patch.bz2

解压缩这两个补丁

bunzip2 utf8-kernel-2.6.22.5-core-1.patch.bz2

bunzip2 utf8-kernel-2.6.22.5-fonts-1.patch.bz2

Chroot到目标系统的目录下，以便不受主系统的影响来制作目标系统

代码:

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

注意：这个时候你的提示符会是“I have no name!”，没有关系继续我们的工作很快就可以正常了。

这个时候如果你关机或重新启动，那么重新开机后回到工作状态的步骤是：

1.重新启动计算机，并从LiveCD启动

2.加载分区

export LFS=/mnt/lfs

mkdir -pv $LFS

mount /dev/hda2 $LFS

3.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

swapon /dev/hda1

4.加载必要的文件系统

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

mount -vt devpts devpts $LFS/dev/pts

mount -vt tmpfs shm $LFS/dev/shm

mount -vt proc proc $LFS/proc

mount -vt sysfs sysfs $LFS/sys

5.Chroot到目标系统下

chroot "$LFS" /tools/bin/env -i \

HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \

/tools/bin/bash --login +h

建立目标系统的目录结构

代码:

```
mkdir -pv /{bin,boot,etc/opt,home,lib,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/{doc,info,locale,man}
mkdir -pv /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
```

```
for dir in /usr /usr/local; do
ln -sv share/{man,doc,info} $dir
done
mkdir -pv /var/{lock,log,mail,run,spool}
mkdir -pv /var/{opt,cache,lib/{misc,locate},local}
```

创建几个必要的链接，因为在目标系统的编译过程中，部分编译程序会用绝对路径来寻找命令或文件。

代码:

```
ln -sv /tools/bin/{bash,cat,echo,grep,pwd,stty} /bin
ln -sv /tools/bin/perl /usr/bin
ln -sv /tools/lib/libgcc_s.so{,.1} /usr/lib
ln -sv /tools/lib/libstdc++.so{,.6} /usr/lib
ln -sv bash /bin/sh
touch /etc/mtab
```

创建root及nobody用户和必要的组

代码:

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
EOF
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape: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:
mail:x:34:
nogroup:x:99:
EOF
```

重新加载bash，以使root用户起效，这样前面的提示符就不会是“I have no name!”

代码:

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

创建和设置几个临时文件和日志文件。

代码:

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

到目前为止，创建目标系统的准备工作以基本完成，下面就要开始目标系统的软件包安装了。

首先进入到源码目录下。

代码:

```
cd /sources
```

（此前已经将LFS需要的源码包加载到了/cdrom下，因此后面的命令将从/cdrom/lfs-sources目录下解出，如果你将源码包直接复制到了sources目录下或别的什么目录下，则要相应的修改下面的命令）

export LFS=/cdrom/lfs-sources

如果之前是将所有源码包复制到sources下的，则执行

代码:

```
export LFS=/sources
```

从现在开始一直到第六章的Stripping Again之前，这个阶段如果你关机或重新启动，那么重新开机后回到工作状态的步骤是：

1.重新启动计算机，并从LiveCD启动

2.加载分区

export LFS=/mnt/lfs

mkdir -pv $LFS

mount /dev/hda2 $LFS

3.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

swapon /dev/hda1

4.加载必要的文件系统

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

mount -vt devpts devpts $LFS/dev/pts

mount -vt tmpfs shm $LFS/dev/shm

mount -vt proc proc $LFS/proc

mount -vt sysfs sysfs $LFS/sys

5.Chroot到目标系统下

chroot "$LFS" /tools/bin/env -i \

HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \

/bin/bash --login +h

6.进入编译目录

cd /sources

export LFS=/sources

Linux-2.6.22.5

代码:

```
tar xvf $LFS/linux-2.6.22.5.tar.bz2
cd linux-2.6.22.5
sed -i '/scsi/d' include/Kbuild
make mrproper
make headers_check
make INSTALL_HDR_PATH=dest headers_install
cp -rv dest/include/* /usr/include
cd ..
rm -rf linux-2.6.22.5
```

Man-pages-2.63

代码:

```
tar xvf $LFS/man-pages-2.63.tar.bz2
cd man-pages-2.63
make install
cd ..
rm -rf man-pages-2.63
```

man-pages的版本可以使用更新的版本。

Glibc-2.5.1

在进行之前请检查一下是否glibc-2.5.1和glibc-build这两个目录已经被删除，如果没有删除请删除后在继续。

代码:

```
tar xvf $LFS/glibc-2.5.1.tar.bz2
cd glibc-2.5.1
tar -xvf $LFS/glibc-libidn-2.5.1.tar.gz
mv glibc-libidn-2.5.1 libidn
sed -i '/vi_VN.TCVN/d' localedata/SUPPORTED
sed -i \
's|libs -o|libs -L/usr/lib -Wl,-dynamic-linker=/lib/ld-linux.so.2 -o|' \
   scripts/test-installation.pl
sed -i 's|@BASH@|/bin/bash|' elf/ldd.bash.in
mkdir -v ../glibc-build
cd ../glibc-build
../glibc-2.5.1/configure --prefix=/usr \
--disable-profile --enable-add-ons \
--enable-kernel=2.6.0 --libexecdir=/usr/lib/glibc
make
make -k check 2>&1 | tee glibc-check-log
grep Error glibc-check-log
touch /etc/ld.so.conf
make install
make localedata/install-locales
```

make localedata/install-locales将安装全部的locale，如果你不想装这么多locale的话就用localedef命令来安装，LFS手册上有例子，如果仅想加入中文的locale，就用

```
mkdir -pv /usr/lib/locale
localedef -i zh_CN -f GB18030 zh_CN
localedef -i zh_CN -f GBK zh_CN
localedef -i zh_CN -f UTF-8 zh_CN
localedef -i zh_CN -f GB2312 zh_CN
localedef -i zh_HK -f UTF-8 zh_CN
localedef -i zh_HK -f BIG5-HKSCS zh_CN
localedef -i zh_TW -f EUC-TW zh_CN
localedef -i zh_TW -f UTF-8 zh_CN
localedef -i zh_TW -f BIG5 zh_CN
```

建立几个重要文件：

代码:

```
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
cp -v --remove-destination /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
cat > /etc/ld.so.conf << "EOF"
/usr/local/lib
/opt/lib
EOF
```

删除编译目录

代码:

```
cd ..
rm -rf glibc-build
rm -rf glibc-2.5.1
```

相关知识：

glibc的测试比较容易出现错误，比如机器慢就有可能出现超时的错误，还有一些能引起错误的LFS手册上有所提及，像超时这种错误有时候很难避免，就跳过去吧。

调整工具链

代码:

```
mv -v /tools/bin/{ld,ld-old}
mv -v /tools/$(gcc -dumpmachine)/bin/{ld,ld-old}
mv -v /tools/bin/{ld-new,ld}
ln -sv /tools/bin/ld /tools/$(gcc -dumpmachine)/bin/ld
```

调整specs文件：

代码:

```
gcc -dumpspecs | sed \
-e 's@/tools/lib/ld-linux.so.2@/lib/ld-linux.so.2@g' \
-e '/\*startfile_prefix_spec:/{n;s@.*@/usr/lib/ @}' \
-e '/\*cpp:/{n;s@$@ -isystem /usr/include@}' > \
`dirname $(gcc --print-libgcc-file-name)`/specs
```

测试工具链是否被调整成功

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

如果显示[Requesting program interpreter: /lib/ld-linux.so.2]表示调整成功，动态库已经连接到了目标系统的/lib下。

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

应该显示

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

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

应该显示

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

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

应该显示

```
SEARCH_DIR("/tools/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/lib")
SEARCH_DIR("/lib");
```

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

应该显示

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

```
grep found dummy.log
```

应该显示

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

如果上面均显示正常，那么表明工具链调整成功，可以删除测试文件了

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

Binutils-2.17

代码:

```
tar xvf $LFS/binutils-2.17.tar.bz2
mkdir binutils-build
cd binutils-build
../binutils-2.17/configure --prefix=/usr --enable-shared
make tooldir=/usr
```

测试：

代码:

```
make check
```

这里测试统计可能会出现个别失败。

代码:

```
make tooldir=/usr install
cp -v ../binutils-2.17/include/libiberty.h /usr/include
cd ..
rm -rf binutils-build
rm -rf binutils-2.17
```

GCC-4.1.2

代码:

```
tar xvf $LFS/gcc-4.1.2.tar.bz2
cd gcc-4.1.2
sed -i 's/install_to_$(INSTALL_DEST) //' libiberty/Makefile.in
sed -i 's/^XCFLAGS =$/& -fomit-frame-pointer/' gcc/Makefile.in
sed -i 's@\./fixinc\.sh@-c true@' gcc/Makefile.in
sed -i 's/@have_mktemp_command@/yes/' gcc/gccbug.in
mkdir -v ../gcc-build
cd ../gcc-build
../gcc-4.1.2/configure --prefix=/usr \
--libexecdir=/usr/lib --enable-shared \
--enable-threads=posix --enable-__cxa_atexit \
--enable-clocale=gnu --enable-languages=c,c++
make
```

测试:

代码:

```
make -k check
```

这里check时间比较长，可能会有一些错误发生

代码:

```
make install
ln -sv ../usr/bin/cpp /lib
ln -sv gcc /usr/bin/cc
cd ..
rm -rf gcc-build
rm -rf gcc-4.1.2
```

重新测试工具链是否正确，确定GCC是否安装正确

echo 'main(){}' > dummy.c

cc dummy.c -v -Wl,--verbose &> dummy.log

readelf -l a.out | grep 'lib'

如果显示[Requesting program interpreter: /lib/ld-linux.so.2]表示链接位置正确，动态库已经连接到了目标系统的/lib下。

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

应该显示

```
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/../../../crt1.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/../../../crti.o succeeded
/usr/lib/gcc/i686-pc-linux-gnu/4.1.2/../../../crtn.o succeeded
```

grep -B3 '^ /usr/include' dummy.log

应该显示

```
#include <...> search starts here:
 /usr/local/include
 /usr/lib/gcc/i686-pc-linux-gnu/4.1.2/include
 /usr/include
```

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

应该显示

```
SEARCH_DIR("/usr/i686-pc-linux-gnu/lib")
SEARCH_DIR("/usr/local/lib")
SEARCH_DIR("/lib")
```

SEARCH_DIR("/usr/lib");
grep "/lib/libc.so.6 " dummy.log
应该显示
attempt to open /lib/libc.so.6 succeeded
grep found dummy.log
应该显示
found ld-linux.so.2 at /lib/ld-linux.so.2
如果上面均显示正常，那么表明工具链正常，可以删除测试文件了
rm -v dummy.c a.out dummy.log

Berkeley DB-4.5.20

代码:

```
tar xvf $LFS/db-4.5.20.tar.gz
cd db-4.5.20
patch -Np1 -i $LFS/db-4.5.20-fixes-1.patch
cd build_unix
../dist/configure --prefix=/usr --enable-compat185 --enable-cxx
make
make docdir=/usr/share/doc/db-4.5.20 install
chown -Rv root:root /usr/share/doc/db-4.5.20
cd /sources
rm -rf db-4.5.20
```

Sed-4.1.5

代码:

```
tar xvf $LFS/sed-4.1.5.tar.gz
cd sed-4.1.5
./configure --prefix=/usr --bindir=/bin --enable-html
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf sed-4.1.5
```

E2fsprogs-1.40.2

代码:

```
tar xvf $LFS/e2fsprogs-1.40.2.tar.gz
cd e2fsprogs-1.40.2
sed -i -e 's@/bin/rm@/tools&@' lib/blkid/test_probe.in
mkdir -v build
cd build
../configure --prefix=/usr --with-root-prefix="" \
--enable-elf-shlibs
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
```

```
make install-libs
cd ../..
rm -rf e2fsprogs-1.40.2
```

Coreutils-6.9

代码:

```
tar xvf $LFS/coreutils-6.9.tar.bz2
cd coreutils-6.9
patch -Np1 -i $LFS/coreutils-6.9-uname-1.patch
patch -Np1 -i $LFS/coreutils-6.9-suppress_uptime_kill_su-1.patch
patch -Np1 -i $LFS/coreutils-6.9-i18n-1.patch
chmod +x tests/sort/sort-mb-tests
./configure --prefix=/usr
make
```

测试：

代码:

```
make NON_ROOT_USERNAME=nobody check-root
echo "dummy:x:1000:nobody" >> /etc/group
su-tools nobody -s /bin/bash -c "make RUN_EXPENSIVE_TESTS=yes check"
```

这个测试应该能正常结束。

删除测试用数据：

代码:

```
sed -i '/dummy/d' /etc/group
```

代码:

```
make install
mv -v /usr/bin/{cat,chgrp,chmod,chown,cp,date,dd,df,echo} /bin
mv -v /usr/bin/{false,hostname,ln,ls,mkdir,mknod,mv,pwd,readlink,rm} /bin
mv -v /usr/bin/{rmdir,stty,sync,true,uname} /bin
mv -v /usr/bin/chroot /usr/sbin
mv -v /usr/bin/{head,sleep,nice} /bin
cd ..
rm -rf coreutils-6.9
```

Iana-Etc-2.20

代码:

```
tar xvf $LFS/iana-etc-2.20.tar.bz2
cd iana-etc-2.20
make
make install
cd ..
rm -rf iana-etc-2.20
```

M4-1.4.10

代码:

```
tar xvf $LFS/m4-1.4.10.tar.bz2
cd m4-1.4.10
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
```

```
cd ..
rm -rf m4-1.4.10
```

Bison-2.3

代码:

```
tar xvf $LFS/bison-2.3.tar.bz2
cd bison-2.3
./configure --prefix=/usr
echo '#define YYENABLE_NLS 1' >> config.h
make
```

测试:

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf bison-2.3
```

Ncurses-5.6

代码:

```
tar xvf $LFS/ncurses-5.6.tar.gz
cd ncurses-5.6
patch -Np1 -i $LFS/ncurses-5.6-coverity_fixes-1.patch
./configure --prefix=/usr --with-shared --without-debug --enable-widec
make
make install
chmod -v 644 /usr/lib/libncurses++w.a
mv -v /usr/lib/libncursesw.so.5* /lib
ln -sfv ../../lib/libncursesw.so.5 /usr/lib/libncursesw.so
for lib in curses ncurses form panel menu ; do \
rm -vf /usr/lib/lib${lib}.so ; \
echo "INPUT(-l${lib}w)" >/usr/lib/lib${lib}.so ; \
ln -sfv lib${lib}w.a /usr/lib/lib${lib}.a ; \
done
ln -sfv libncurses++w.a /usr/lib/libncurses++.a
rm -vf /usr/lib/libcursesw.so
echo "INPUT(-lncursesw)" >/usr/lib/libcursesw.so
ln -sfv libncurses.so /usr/lib/libcurses.so
ln -sfv libncursesw.a /usr/lib/libcursesw.a
ln -sfv libncurses.a /usr/lib/libcurses.a
cd ..
rm -rf ncurses-5.6
```

Procps-3.2.7

代码:

```
tar xvf $LFS/procps-3.2.7.tar.gz
cd procps-3.2.7
make
make install
cd ..
rm -rf procps-3.2.7
```

Libtool-1.5.24

代码:

```
tar xvf $LFS/libtool-1.5.24.tar.gz
cd libtool-1.5.24
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf libtool-1.5.24
```

Perl-5.8.8

代码:

```
tar xvf $LFS/perl-5.8.8.tar.bz2
cd perl-5.8.8
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
./configure.gnu --prefix=/usr \
    -Dman1dir=/usr/share/man/man1 \
    -Dman3dir=/usr/share/man/man3 \
    -Dpager="/usr/bin/less -isR"
make
```

测试：

代码:

```
make test
```

这里test不会有错

代码:

```
make install
cd ..
rm -rf perl-5.8.8
```

Readline-5.2

代码:

```
tar xvf $LFS/readline-5.2.tar.gz
cd readline-5.2
sed -i '/MV.*old/d' Makefile.in
sed -i '/{OLDSUFF}/c:' support/shlib-install
patch -Np1 -i $LFS/readline-5.2-fixes-3.patch
./configure --prefix=/usr --libdir=/lib
make SHLIB_XLDFLAGS=-lncurses
make install
mv -v /lib/lib{readline,history}.a /usr/lib
rm -v /lib/lib{readline,history}.so
ln -sfv ../../lib/libreadline.so.5 /usr/lib/libreadline.so
ln -sfv ../../lib/libhistory.so.5 /usr/lib/libhistory.so
cd ..
rm -rf readline-5.2
```

Zlib-1.2.3

代码:

```
tar xvf $LFS/zlib-1.2.3.tar.gz
cd zlib-1.2.3
./configure --prefix=/usr --shared --libdir=/lib
make
```

测试动态链接库：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
rm -v /lib/libz.so
ln -sfv ../../lib/libz.so.1.2.3 /usr/lib/libz.so
make clean
./configure --prefix=/usr
make
```

测试静态库

代码:

```
make check
```

这里check不会有错

代码:

```
make install
chmod -v 644 /usr/lib/libz.a
cd ..
rm -rf zlib-1.2.3
```

Autoconf-2.61

代码:

```
tar xvf $LFS/autoconf-2.61.tar.bz2
cd autoconf-2.61
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里测试时间比较长，但不会有错

代码:

```
make install
cd ..
rm -rf autoconf-2.61
```

Automake-1.10

代码:

```
tar xvf $LFS/automake-1.10.tar.bz2
cd automake-1.10
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里测试时间比较长，可能会有3个测试失败，但能顺利结束。

代码:

```
make install
cd ..
rm -rf automake-1.10
```

Bash-3.2

代码:

```
tar xvf $LFS/bash-3.2.tar.gz
cd bash-3.2
tar -xvf $LFS/bash-doc-3.2.tar.gz &&
sed -i "s|htmldir = @htmldir@|htmldir = /usr/share/doc/bash-3.2|" Makefile.in
patch -Np1 -i $LFS/bash-3.2-fixes-5.patch
./configure --prefix=/usr --bindir=/bin --without-bash-malloc --with-installed-readline
make
```

测试：

代码:

```
sed -i 's/LANG/LC_ALL/' tests/intl.tests
sed -i 's@tests@& </dev/tty@' tests/run-test
chown -Rv nobody ./
su-tools nobody -s /bin/bash -c "make tests"
```

这里check不会有错，可能会有不少警告。

代码:

```
make install
cd ..
rm -rf bash-3.2
```

应用刚编译好的/bin/bash：

代码:

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

Bzip2-1.0.4

代码:

```
tar xvf $LFS/bzip2-1.0.4.tar.gz
cd bzip2-1.0.4
patch -Np1 -i $LFS/bzip2-1.0.4-install_docs-1.patch
make -f Makefile-libbz2_so
make clean
make
make PREFIX=/usr install
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
cd ..
rm -rf bzip2-1.0.4
```

Diffutils-2.8.1

代码:

```
tar xvf $LFS/diffutils-2.8.1.tar.gz
cd diffutils-2.8.1
patch -Np1 -i $LFS/diffutils-2.8.1-i18n-1.patch
touch man/diff.1
./configure --prefix=/usr
make
make install
cd ..
rm -rf diffutils-2.8.1
```

File-4.21

代码:

```
tar xvf $LFS/file-4.21.tar.gz
```

```
cd file-4.21
./configure --prefix=/usr
make
make install
cd ..
rm -rf file-4.21
```

Findutils-4.2.31

代码:

```
tar xvf $LFS/findutils-4.2.31.tar.gz
cd findutils-4.2.31
./configure --prefix=/usr --libexecdir=/usr/lib/findutils \
--localstatedir=/var/lib/locate
make
```

测试:

代码:

```
make check
```

这里check不会有错

代码:

```
make install
mv -v /usr/bin/find /bin
sed -i -e 's/find:=${BINDIR}/find:=\/bin/' /usr/bin/updatedb
cd ..
rm -rf findutils-4.2.31
```

Flex-2.5.33

代码:

```
tar xvf $LFS/flex-2.5.33.tar.bz2
cd flex-2.5.33
./configure --prefix=/usr
make
```

测试:

代码:

```
make check
```

这里check不会有错

代码:

```
make install
ln -sv libfl.a /usr/lib/libl.a
```

创建一个lex的命令。

代码:

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Begin /usr/bin/lex
exec /usr/bin/flex -l "$@"
# End /usr/bin/lex
EOF
chmod -v 755 /usr/bin/lex
```

代码:

```
cd ..
rm -rf flex-2.5.33
```

GRUB-0.97

代码:

```
tar xvf $LFS/grub-0.97.tar.gz
```

```
cd grub-0.97
patch -Np1 -i $LFS/grub-0.97-disk_geometry-1.patch
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
mkdir -v /boot/grub
cp -v /usr/lib/grub/i386-pc/stage{1,2} /boot/grub
cd ..
rm -rf grub-0.97
```

Gawk-3.1.5

代码:

```
tar xvf $LFS/gawk-3.1.5.tar.bz2
cd gawk-3.1.5
patch -Np1 -i $LFS/gawk-3.1.5-segfault_fix-1.patch
./configure --prefix=/usr --libexecdir=/usr/lib
cat >> config.h << "EOF"
#define HAVE_LANGINFO_CODESET 1
#define HAVE_LC_MESSAGES 1
EOF
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf gawk-3.1.5
```

Gettext-0.16.1

代码:

```
tar xvf $LFS/gettext-0.16.1.tar.gz
cd gettext-0.16.1
./configure --prefix=/usr
make
```

测试：

代码:

```
make check
```

这里check的时间比较长，但不会有错。

代码:

```
make install
cd ..
rm -rf gettext-0.16.1
```

Grep-2.5.1a

代码:

```
tar xvf $LFS/grep-2.5.1a.tar.bz2
cd grep-2.5.1a
```

```
patch -Np1 -i $LFS/grep-2.5.1a-redhat_fixes-2.patch
chmod +x tests/fmbtest.sh
./configure --prefix=/usr --bindir=/bin
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf grep-2.5.1a
```

Groff-1.18.1.4

代码:

```
tar xvf $LFS/groff-1.18.1.4.tar.gz
cd groff-1.18.1.4
patch -Np1 -i $LFS/groff-1.18.1.4-debian_fixes-1.patch
sed -i -e 's/2010/002D/' -e 's/2212/002D/' \
-e 's/2018/0060/' -e 's/2019/0027/' font/devutf8/R.proto
PAGE=A4 ./configure --prefix=/usr --enable-multibyte
make
make install
ln -sv eqn /usr/bin/geqn
ln -sv tbl /usr/bin/gtbl
cd ..
rm -rf groff-1.18.1.4
```

Gzip-1.3.12

代码:

```
tar xvf $LFS/gzip-1.3.12.tar.gz
cd gzip-1.3.12
./configure --prefix=/usr --bindir=/bin
make
```

测试：

代码:

```
make check
```

这里check不会有错

代码:

```
make install
mv -v /bin/{gzexe,uncompress,zcmp,zdiff,zegrep} /usr/bin
mv -v /bin/{zfgrep,zforce,zgrep,zless,zmore,znew} /usr/bin
cd ..
rm -rf gzip-1.3.12
```

Inetutils-1.5

代码:

```
tar xvf $LFS/inetutils-1.5.tar.gz
cd inetutils-1.5
patch -Np1 -i $LFS/inetutils-1.5-no_server_man_pages-2.patch
./configure --prefix=/usr --libexecdir=/usr/sbin \
--sysconfdir=/etc --localstatedir=/var \
--disable-ifconfig --disable-logger --disable-syslogd \
--disable-whois --disable-servers
make
```

```
make install
mv -v /usr/bin/ping /bin
cd ..
rm -rf inetutils-1.5
```

IPRoute2-2.6.20-070313

代码:

```
tar xvf $LFS/iproute2-2.6.20-070313.tar.bz2
cd iproute-2.6.20-070313
sed -i -e '/tc-bfifo.8/d' -e '/tc-pfifo.8/s/pbfifo/bfifo/' Makefile
make SBINDIR=/sbin
make SBINDIR=/sbin install
mv -v /sbin/arpd /usr/sbin
cd ..
rm -rf iproute-2.6.20-070313
```

Kbd-1.12

代码:

```
tar xvf $LFS/kbd-1.12.tar.bz2
cd kbd-1.12
patch -Np1 -i $LFS/kbd-1.12-backspace-1.patch
patch -Np1 -i $LFS/kbd-1.12-gcc4_fixes-1.patch
./configure --datadir=/lib/kbd
make
make install
mv -v /usr/bin/{kbd_mode,openvt,setfont} /bin
cd ..
rm -rf kbd-1.12
```

Less-406

代码:

```
tar xvf $LFS/less-406.tar.gz
cd less-406
./configure --prefix=/usr --sysconfdir=/etc
make
make install
cd ..
rm -rf less-406
```

Make-3.81

代码:

```
tar xvf $LFS/make-3.81.tar.bz2
cd make-3.81
./configure --prefix=/usr
make
```

测试:

代码:

```
make check
```

这里check不会有错

代码:

```
make install
cd ..
rm -rf make-3.81
```

Man-DB-2.4.4

代码:

```
tar xvf $LFS/man-db-2.4.4.tar.gz
cd man-db-2.4.4
mv man/de{_DE.88591,}
mv man/es{_ES.88591,}
mv man/it{_IT.88591,}
sed -i 's,\*_\*,??,' man/Makefile.in
sed -i -e '\%\t/usr/man%d' -e '\%\t/usr/local/man%d' src/man_db.conf.in
cat >> include/manconfig.h.in << "EOF"
#define WEB_BROWSER "exec /usr/bin/lynx"
#define COL "/usr/bin/col"
#define VGRIND "/usr/bin/vgrind"
#define GRAP "/usr/bin/grap"
EOF
patch -Np1 -i $LFS/man-db-2.4.4-fixes-1.patch
./configure --prefix=/usr --enable-mb-groff --disable-setuid
make
make install
```

创建一个用于转换man手册编码的脚本

代码:

```
cat >> convert-mans << "EOF"
#!/bin/sh -e
FROM="$1"
TO="$2"
shift ; shift
while [ $# -gt 0 ]
do
   FILE="$1"
   shift
   iconv -f "$FROM" -t "$TO" "$FILE" >.tmp.iconv
   mv .tmp.iconv "$FILE"
done
EOF
```

代码:

```
install -m755 convert-mans  /usr/bin
cd ..
rm -rf man-db-2.4.4
```

Mktemp-1.5

代码:

```
tar xvf $LFS/mktemp-1.5.tar.gz
cd mktemp-1.5
patch -Np1 -i $LFS/mktemp-1.5-add_tempfile-3.patch
./configure --prefix=/usr --with-libc
make
make install
make install-tempfile
cd ..
rm -rf mktemp-1.5
```

Module-Init-Tools-3.2.2

代码:

```
tar xvf $LFS/module-init-tools-3.2.2.tar.bz2
cd module-init-tools-3.2.2
patch -Np1 -i $LFS/module-init-tools-3.2.2-modprobe-1.patch
./configure
```

```
make check
make distclean
./configure --prefix=/ --enable-zlib
make
make INSTALL=install install
cd ..
rm -rf module-init-tools-3.2.2
```

Patch-2.5.4

代码:

```
tar xvf $LFS/patch-2.5.4.tar.gz
cd patch-2.5.4
./configure --prefix=/usr
make
make install
cd ..
rm -rf patch-2.5.4
```

Psmisc-22.5

代码:

```
tar xvf $LFS/psmisc-22.5.tar.gz
cd psmisc-22.5
./configure --prefix=/usr --exec-prefix=""
make
make install
mv -v /bin/pstree* /usr/bin
ln -sv killall /bin/pidof
cd ..
rm -rf psmisc-22.5
```

Shadow-4.0.18.1

代码:

```
tar xvf $LFS/shadow-4.0.18.1.tar.bz2
cd shadow-4.0.18.1
patch -Np1 -i $LFS/shadow-4.0.18.1-useradd_fix-2.patch
./configure --libdir=/lib --sysconfdir=/etc --enable-shared --without-selinux
sed -i 's/groups$(EXEEXT) //' src/Makefile
find man -name Makefile -exec sed -i 's/groups\.1 / /' {} \;
sed -i -e 's@#MD5_CRYPT_ENAB.no@MD5_CRYPT_ENAB yes@' \
-e 's@/var/spool/mail@/var/mail@' etc/login.defs
make
make install
mv -v /usr/bin/passwd /bin
mv -v /lib/libshadow.*a /usr/lib
rm -v /lib/libshadow.so
ln -sfv ../../lib/libshadow.so.0 /usr/lib/libshadow.so
```

配置增加用户的默认设置。

代码:

```
pwconv
grpconv
useradd -D -b /home
sed -i 's/yes/no/' /etc/default/useradd
```

设置root用户密码:

代码:

```
passwd root
```

一定要设置root的密码，否则重新启动后无法登陆。

代码:

```
cd ..
rm -rf shadow-4.0.18.1
```

Sysklogd-1.4.1

代码:

```
tar xvf $LFS/sysklogd-1.4.1.tar.gz
cd sysklogd-1.4.1
patch -Np1 -i $LFS/sysklogd-1.4.1-fixes-2.patch
patch -Np1 -i $LFS/sysklogd-1.4.1-8bit-1.patch
make
make install
```

设置syslog的配置文件

代码:

```
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
```

代码:

```
cd ..
rm -rf sysklogd-1.4.1
```

Sysvinit-2.86

代码:

```
tar xvf $LFS/sysvinit-2.86.tar.gz
cd sysvinit-2.86
sed -i 's@Sending processes@& configured via /etc/inittab@g' src/init.c
make -C src
make -C src install
```

设置启动配置文件。

代码:

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab
id:3:initdefault:
si::sysinit:/etc/rc.d/init.d/rc sysinit
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 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
```

代码:

```
cd ..
rm -rf sysvinit-2.86
```

Tar-1.18

代码:

```
tar xvf $LFS/tar-1.18.tar.bz2
cd tar-1.18
./configure --prefix=/usr --bindir=/bin --libexecdir=/usr/sbin
make
```

测试:

代码:

```
make check
```

这里check应能通过

代码:

```
make install
cd ..
rm -rf tar-1.18
```

Texinfo-4.9

代码:

```
tar xvf $LFS/texinfo-4.9.tar.bz2
cd texinfo-4.9
patch -Np1 -i $LFS/texinfo-4.9-multibyte-1.patch
patch -Np1 -i $LFS/texinfo-4.9-tempfile_fix-1.patch
./configure --prefix=/usr
make
```

测试:

代码:

```
make check
```

这里check不会有错

代码:

```
make install
make TEXMF=/usr/share/texmf install-tex
```

代码:

```
cd /usr/share/info
rm dir
for f in *
do install-info $f dir 2>/dev/null
done
```

代码:

```
cd /sources
rm -rf texinfo-4.9
```

Udev-113

代码:

```
tar xvf $LFS/udev-113.tar.bz2
cd udev-113
```

```
tar -xvf $LFS/udev-config-6.3.tar.bz2
install -dv /lib/{firmware,udev/devices/{pts,shm}}
mknod -m0666 /lib/udev/devices/null c 1 3
ln -sv /proc/self/fd /lib/udev/devices/fd
ln -sv /proc/self/fd/0 /lib/udev/devices/stdin
ln -sv /proc/self/fd/1 /lib/udev/devices/stdout
ln -sv /proc/self/fd/2 /lib/udev/devices/stderr
ln -sv /proc/kcore /lib/udev/devices/core
make EXTRAS="`echo extras/*/`"
```

测试：

代码:

```
make test
```

这里test可能有错误，不必理会。

代码:

```
make DESTDIR=/ EXTRAS="`echo extras/*/`" install
cp -v etc/udev/rules.d/[0-9]* /etc/udev/rules.d/
cd udev-config-6.3
make install
make install-doc
make install-extra-doc
cd ..
install -m644 -v docs/writing_udev_rules/index.html \
/usr/share/doc/udev-113/index.html
cd ..
rm -rf udev-113
```

Util-linux-2.12r

代码:

```
tar xvf $LFS/util-linux-2.12r.tar.bz2
cd util-linux-2.12r
sed -e 's@etc/adjtime@var/lib/hwclock/adjtime@g' \
-i $(grep -rl '/etc/adjtime' .)
mkdir -pv /var/lib/hwclock
patch -Np1 -i $LFS/util-linux-2.12r-cramfs-1.patch
patch -Np1 -i $LFS/util-linux-2.12r-lseek-1.patch
./configure
make HAVE_KILL=yes HAVE_SLN=yes
make HAVE_KILL=yes HAVE_SLN=yes install
cd ..
rm -rf util-linux-2.12r
```

Vim-7.1

代码:

```
tar xvf $LFS/vim-7.1.tar.bz2
tar xvf $LFS/vim-7.1-lang.tar.gz
cd vim71
patch -Np1 -i $LFS/vim-7.1-fixes-1.patch
patch -Np1 -i $LFS/vim-7.1-mandir-1.patch
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
./configure --prefix=/usr --enable-multibyte
make
```

测试：

代码:

```
make test
```

这里test可能会有个别错误

代码:

```
make install
ln -sv vim /usr/bin/vi
for L in "" fr it pl ru; do
ln -sv vim.1 /usr/share/man/$L/man1/vi.1
done
ln -sv ../vim/vim71/doc /usr/share/doc/vim-7.1
```

建立vim的默认配置文件

代码:

```
cat > /etc/vimrc << "EOF"
" Begin /etc/vimrc
set nocompatible
set backspace=2
syntax on
if (&term == "iterm") || (&term == "putty")
set background=dark
endif
" End /etc/vimrc
EOF
```

代码:

```
cd ..
rm -rf vim71
```

退出chroot环境：

代码:

```
logout
```

为Strip而进入chroot环境：

代码:

```
chroot $LFS /tools/bin/env -i \
HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin \
/tools/bin/bash --login
```

如果现在重新启动，那么重新开机后回到工作状态的步骤是：

1.重新启动计算机，并从LiveCD启动

2.加载分区

export LFS=/mnt/lfs

mkdir -pv $LFS

mount /dev/hda2 $LFS

3.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

swapon /dev/hda1

4.加载必要的文件系统

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

mount -vt devpts devpts $LFS/dev/pts

mount -vt tmpfs shm $LFS/dev/shm

mount -vt proc proc $LFS/proc

mount -vt sysfs sysfs $LFS/sys

5.Chroot到目标系统下

chroot "$LFS" /tools/bin/env -i \

HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \

PATH=/bin:/usr/bin:/sbin:/usr/sbin \

/tools/bin/bash --login

Strip

代码:

```
/tools/bin/find /{,usr/}{bin,lib,sbin} -type f \
-exec /tools/bin/strip --strip-debug '{}' ';'
```

退出chroot环境

代码:

```
logout
```

为最后的设置进入chroot环境

代码:

```
chroot "$LFS" /usr/bin/env -i \
HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \
PATH=/bin:/usr/bin:/sbin:/usr/sbin \
/bin/bash --login
cd /sources
export LFS=/sources
```

从现在开始一直到制作结束，重新开机后回到工作状态的步骤是：

1.重新启动计算机，并从LiveCD启动

2.加载分区

export LFS=/mnt/lfs

mkdir -pv $LFS

mount /dev/hda2 $LFS

3.加载交换分区（如果不想用交换分区或者没有交换分区可跳过此步骤）

swapon /dev/hda1

4.加载必要的文件系统

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

mount -vt devpts devpts $LFS/dev/pts

mount -vt tmpfs shm $LFS/dev/shm

mount -vt proc proc $LFS/proc

mount -vt sysfs sysfs $LFS/sys

5.Chroot到目标系统下

chroot "$LFS" /usr/bin/env -i \

HOME=/root TERM="$TERM" PS1='\u:\w\$ ' \

PATH=/bin:/usr/bin:/sbin:/usr/sbin \

/bin/bash --login

6.进入编译目录

cd /sources

export LFS=/sources

LFS-Bootscripts-6.3

代码:

```
tar xvf $LFS/lfs-bootscripts-6.3.tar.bz2
cd lfs-bootscripts-6.3
make install
cd ..
rm -rf lfs-bootscripts-6.3
```

时间设置（Configuring the setclock Script）

代码:

```
cat > /etc/sysconfig/clock << "EOF"
# Begin /etc/sysconfig/clock
UTC=1
# End /etc/sysconfig/clock
EOF
```

设置bash下的键盘功能键设置

代码:

```
cat > /etc/inputrc << "EOF"
set horizontal-scroll-mode Off
set meta-flag On
set input-meta On
set convert-meta Off
set output-meta On
set bell-style none
"\eOd": backward-word
"\eOc": forward-word
"\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
"\eOH": beginning-of-line
"\eOF": end-of-line
"\e[H": beginning-of-line
"\e[F": end-of-line
EOF
```

设置Bash Shell启动文件（The Bash Shell Startup Files）

代码:

```
cat > /etc/profile << "EOF"
# Begin /etc/profile
export LANG=zh_CN.UTF-8
export INPUTRC=/etc/inputrc
alias ls="ls --color"
export PS1='\u:\w\$ '
# End /etc/profile
EOF
```

设置本地网络名

代码:

```
echo "HOSTNAME=mylinux" > /etc/sysconfig/network
```

设置hosts文件

代码:

```
cat > /etc/hosts << "EOF"
# Begin /etc/hosts (no network card version)
127.0.0.1 mylinux localhost
# End /etc/hosts (no network card version)
EOF
```

设置网络的静态地址

代码:

```
cd /etc/sysconfig/network-devices &&
mkdir -v ifconfig.eth0 &&
cat > ifconfig.eth0/ipv4 << "EOF"
ONBOOT=yes
SERVICE=ipv4-static
IP=192.168.1.1
GATEWAY=192.168.1.2
PREFIX=24
BROADCAST=192.168.1.255
EOF
```

注意：IP、GATEWAY、BROADCAST的地址根据自己的实际情况设置。

设置DNS

代码:

```
cat > /etc/resolv.conf << "EOF"
# Begin /etc/resolv.conf
nameserver 你的首个DNS的地址
nameserver 你的第二DNS的地址
# End /etc/resolv.conf
EOF
```

建立fstab文件

代码:

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab
# file system  mount-point  type    options          dump  fsck
#                                                          order
/dev/hda2       /            xfs   defaults         1     1
/dev/hda1       swap         swap    pri=1            0     0
proc              /proc        proc    defaults         0     0
sysfs             /sys         sysfs   defaults         0     0
devpts          /dev/pts     devpts gid=4,mode=620   0     0
shm              /dev/shm     tmpfs   defaults         0     0
# End /etc/fstab
EOF
```

注意：这里的磁盘名以及文件系统名需要根据实际情况修改。

安装内核Linux-2.6.22.5

代码:

```
cd /sources
tar xvf $LFS/linux-2.6.22.5.tar.bz2
cd linux-2.6.22.5
```

安装显示UTF-8编码文字的补丁

如果之前下载了UTF-8编码文字显示补丁，那么这里可以将这两个补丁打入内核

patch -Np1 -i $LFS/utf8-kernel-2.6.22.5-core-1.patch

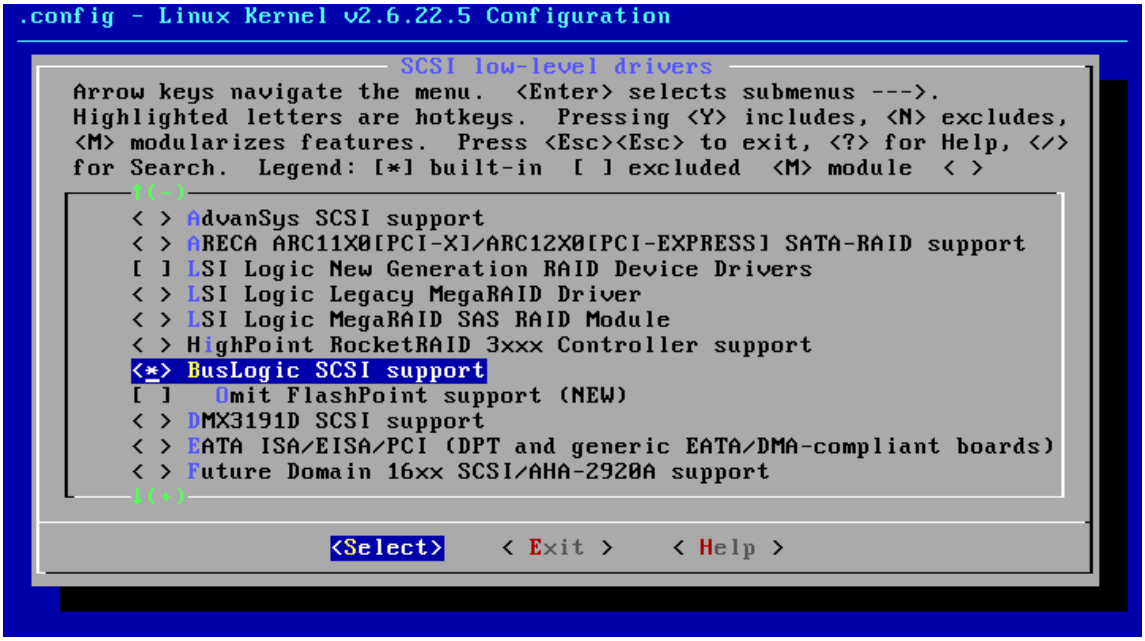
patch -Np1 -i $LFS/utf8-kernel-2.6.22.5-fonts-1.patch

代码:

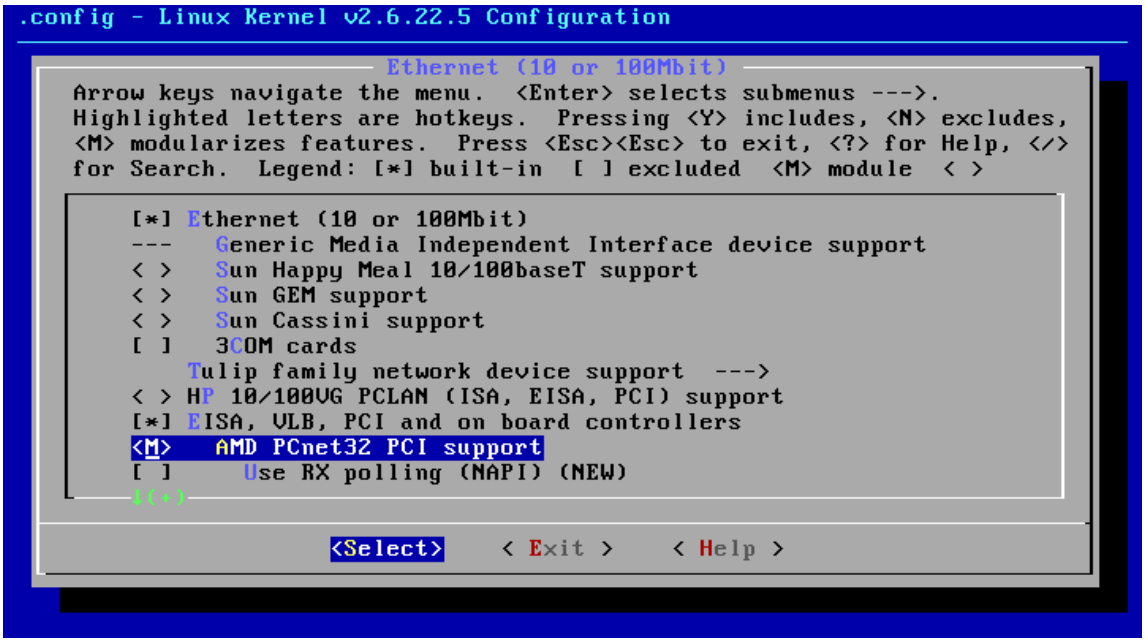
```
make mrproper
make menuconfig
```

根据你的机器实际情况配置内核选项，这里为了说明方便，以VMWare5.5为基础虚拟的硬件来配置内核

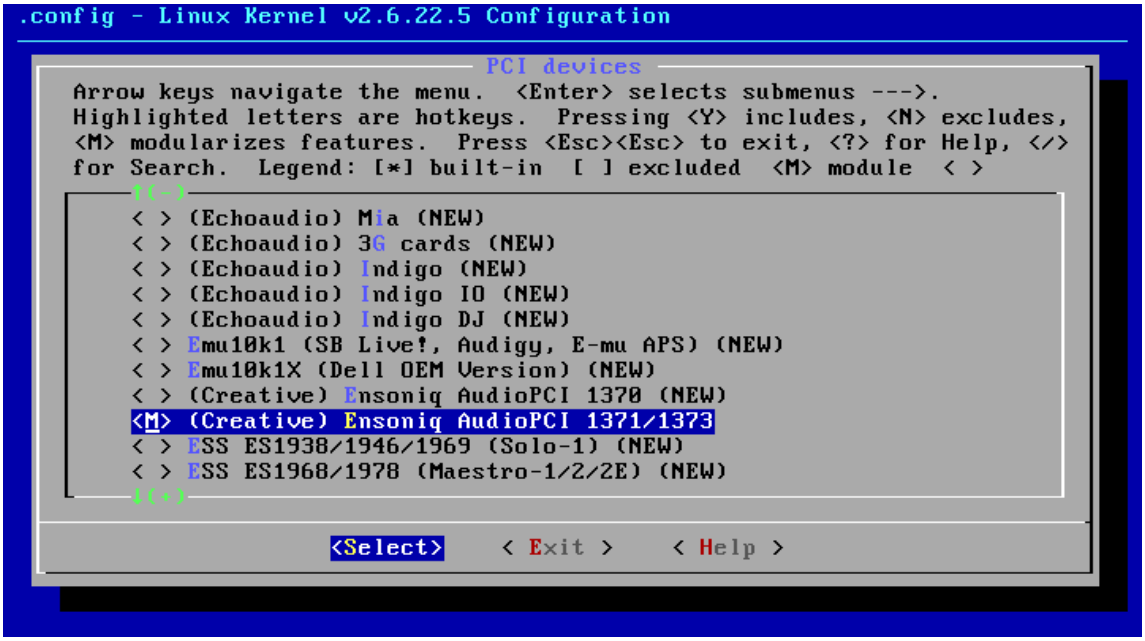
如果在建立虚拟机的时候是选择的BusLogic的SCSI磁盘，那么应该在Device Drivers->SCSI device support->SCSI low-level drivers下加入BusLogic SCSI support的支持，可以采用编译到内核来避免未用initrd脚本来加载模块而导致启动失败



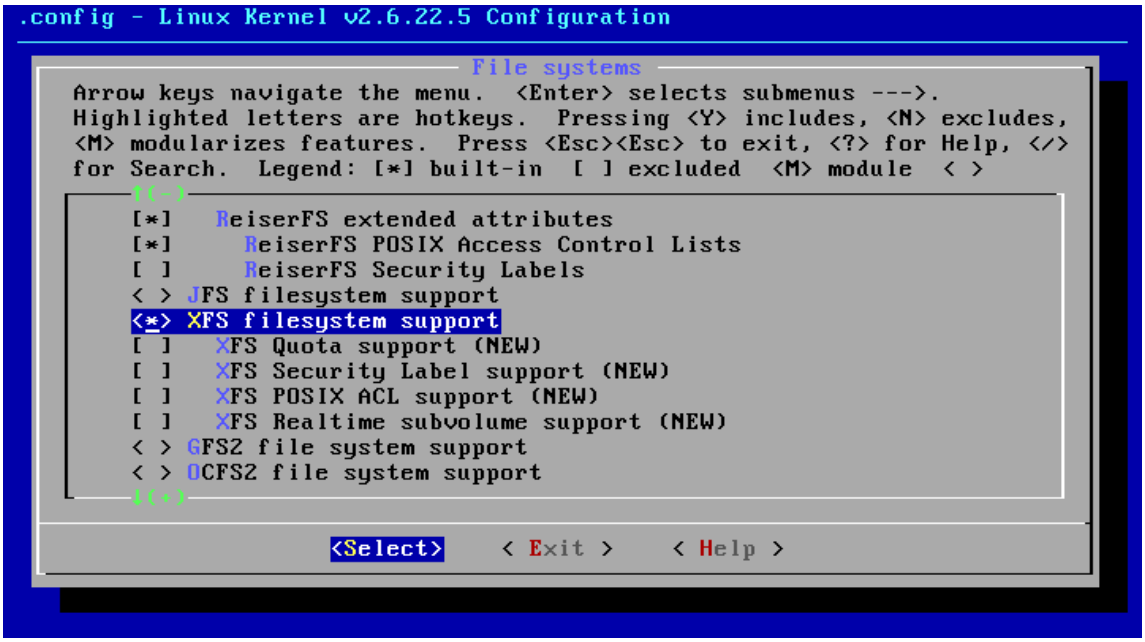
如果想支持网络则应该在Device Drivers->Networking support->Ethernet (10 or 100Mbit)加入AMD PCnet32 PCI support的支持，可以采用编译到内核也可以编译成模块的方式



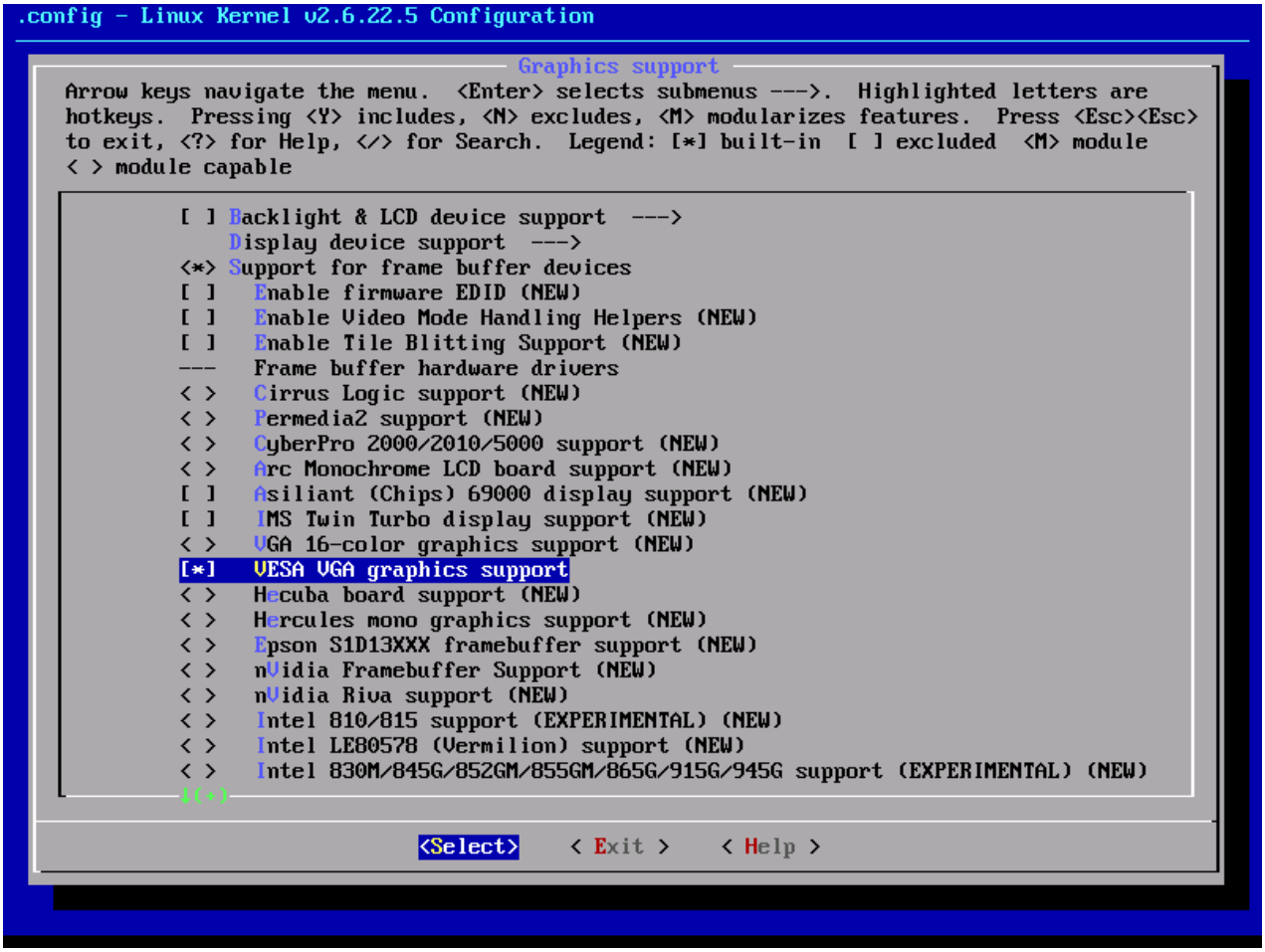
如果想支持声卡则应该在Device Drivers->Sound->Advanced Linux Sound Architecture->PCI devices加入(Creative) Ensoniq AudioPCI 1371/1373的支持，编译成模块即可



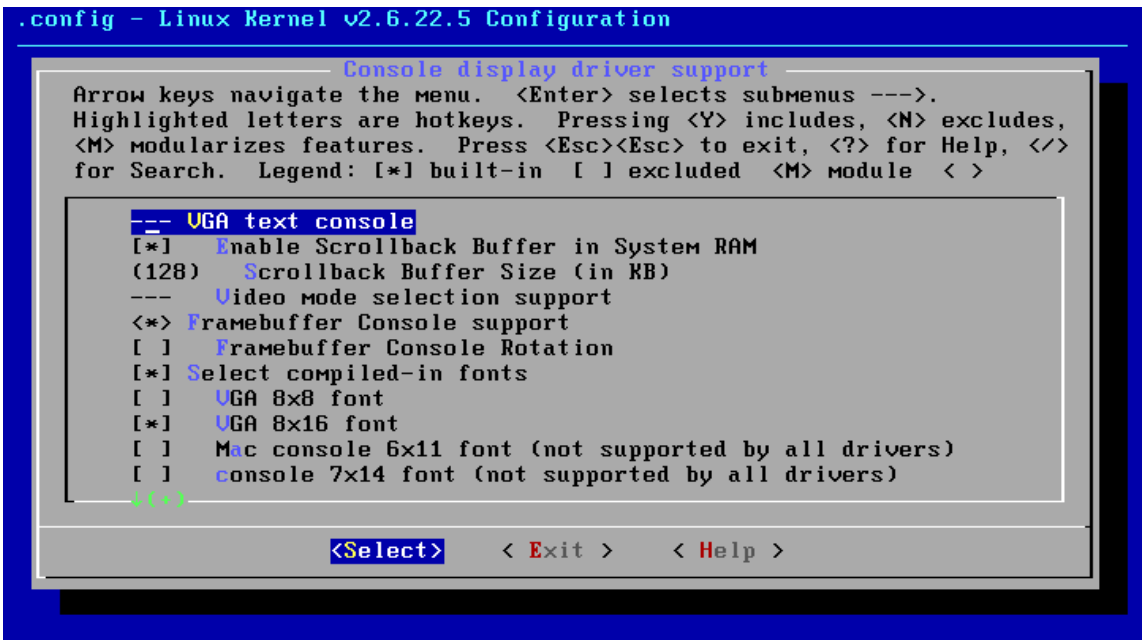
文件系统的支持，需要根据之前将目标系统分区格式化的情况而定，使用了什么文件系统就需要加入该文件系统的支持，因之前采用的是Xfs文件系统，因此在File Systems->XFS support加入XFS的支持，可以采用编译到内核来避免未用initrd脚本来加载模块而导致启动失败



如果之前给内核加入了显示UTF-8编码文字的补丁的话，那么这里需要加入framebuffer的支持才能使补丁生效，在Device Drivers->Graphics support中加入Support for frame buffer devices，并选择上VESA VGA graphics supports，这里将其编译到内核中



同时还需要加入framebuffer字体支持，在Device Drivers->Graphics support->Console display driver support中加入Framebuffer Console support并选择上Select compiled-in fonts，选上VGA 8x16 font这一种字体就可以了，这里将这些选择都编译到内核中



可以保存退出了

代码:

```
make
make modules_install
cp -v arch/i386/boot/bzImage /boot/lfskernel-2.6.22.5
```

```
cp -v System.map /boot/System.map-2.6.22.5
cp -v .config /boot/config-2.6.22.5
install -d /usr/share/doc/linux-2.6.22.5
cp -r Documentation/* /usr/share/doc/linux-2.6.22.5
```

安装Grub，使系统能启动，这里设置需要根据情况而修改，这里以之前介绍的分区设置为例：

代码:

```
grub
```

输入root (hd0,1)

输入setup (hd0)

quit

设置grub启动菜单

代码:

```
cat > /boot/grub/menu.lst << "EOF"
# Begin /boot/grub/menu.lst
# By default boot the first menu entry.
default 0
# Allow 30 seconds before booting the default.
timeout 30
# Use prettier colors.
color green/black light-green/black
# The first entry is for LFS.
title LFS 6.3
root (hd0,1)
kernel /boot/lfskernel-2.6.22.5 root=/dev/hda2 vga=788
EOF
```

注意：这里root后面的磁盘分区需要根据实际情况调整。

将menu.lst连接到/etc目录下

代码:

```
mkdir -v /etc/grub
ln -sv /boot/grub/menu.lst /etc/grub
```

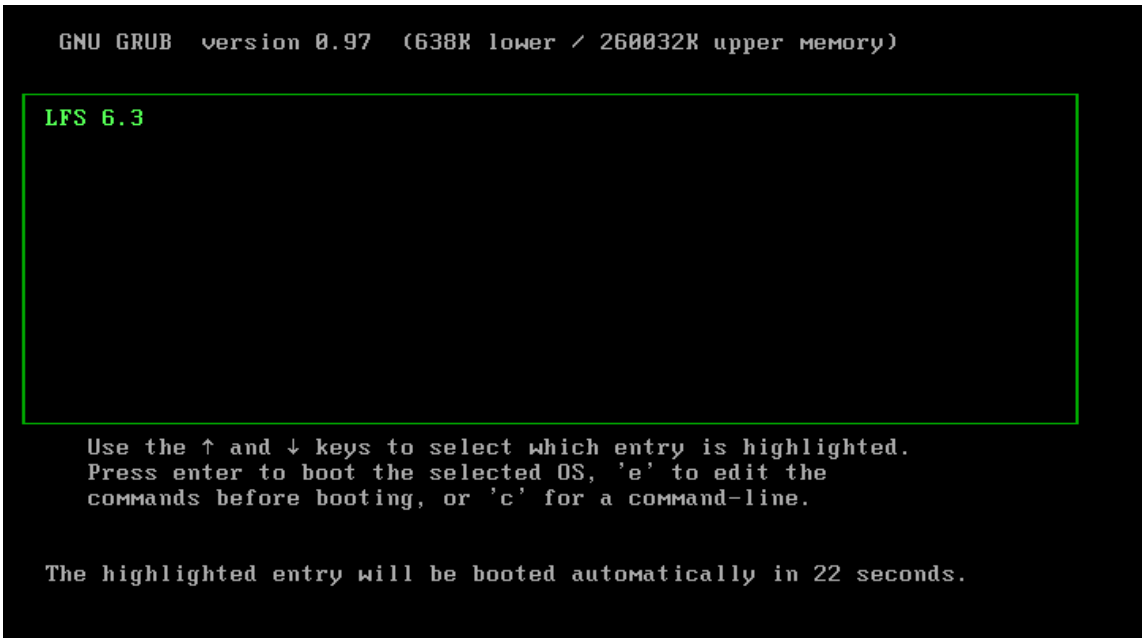
退出制作环境：

代码:

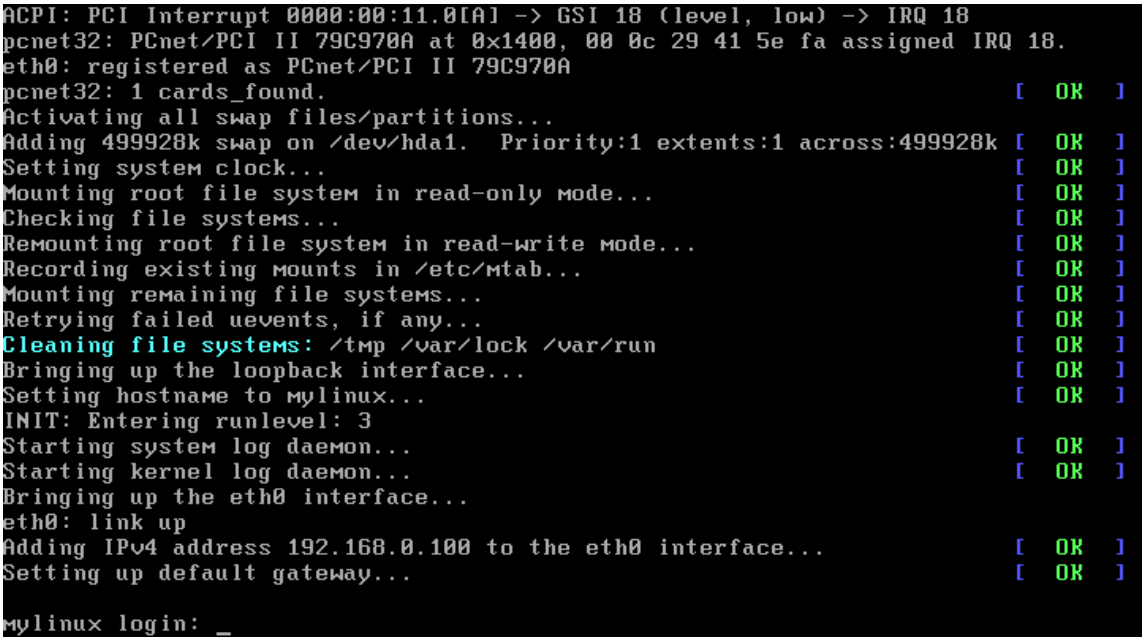
```
logout
```

现在已经完成了lfs的安装，可以重新启动来运行我们自己的系统咯！

Grub启动界面



启动完成



作者：孙海勇（冲天飞豹）
Email:youbest@sina.com
2008年2月10日

更新日志：

2008年2月10日：本文发布。

2008年2月12日：

将chown -R root:root $LFS/tools误写为chwon -R root:root $LFS/tools
已改正过来
由linuxsir上的"糊涂"发现并报告

2008年2月12日：

在目标系统的/etc/profile中增加

alias ls="ls --color"
export PS1='\u:\w\$ '

2008年3月3日：
修改命令
gcc -dumpspecs | sed 's@^/lib/ld-linux.so.2@/tools&@g' \
> `dirname $(gcc -print-libgcc-file-name)`/specs
为
gcc -dumpspecs | sed 's@^/lib/ld-linux.so.2@/tools&@g' > `dirname $(gcc -print-libgcc-file-name)`/specs

原因，为前面的“>”是输入符号，而在命令输入过程中系统会自动出现一个“>”，为避免新手误解，修改该命令表达方式。
由cublog上的“飞天老鼠”指出。

2008年3月3日：
修改命令
./configure.gnu --prefix=/usr \
-Dman1dir=/usr/share/man/man1
-Dman3dir=/usr/share/man/man3
-Dpager="/usr/bin/less -isR"
为
./configure.gnu --prefix=/usr \
-Dman1dir=/usr/share/man/man1 \
-Dman3dir=/usr/share/man/man3 \
-Dpager="/usr/bin/less -isR"
原因为笔误，缺少两个断行符号。
由cublog上的“embeddedarmlinux”指出

2008年3月16日：
修改笔误
rm-rf gcc-3.4.3
改为
rm -rf gcc-4.1.2
由linuxsir上的“yagng”指出

2008年7月6日：
修改笔误
tar xvf /lfs-sources/findutils-4.2.31.tar.bz2
改为
tar xvf /lfs-sources/findutils-4.2.31.tar.gz
由cublog上的“坚硬的贝壳”指出

2008年7月6日：
修改笔误
在第六章中glibc的locale安装部分
mkdir -pv /tools/lib/locale
改为
mkdir -pv /usr/lib/locale
由cublog上的“Leon”指出

2008年7月7日：
增加显示UTF-8编码文字
给内核打上我制作的UTF-8编码文字显示补丁，这样可以让制作出来的系统在带有framebuffer普通的终端下（非X环境）就可以直接显示出中文及其它语言的文字，并且可以同屏显示多国文字。