Build Guide

Linux From Scratch on the Raspberry Pi

This guide will take you through the Pi specific steps of an LFS build.

About The Guide

We're going to work our way through the development version (http://www.linuxfromscratch.org/lfs/view/development/) of LFS, while paying extra attention to the steps where the instructions differ for the Pi. It is therefore recommended that you keep the LFS book open in another browser tab. You'll find a pointer to the corresponding section of the LFS book at each step. The LFS book steps that are not mentioned can be assumed to work on the Pi exactly as printed.

This guide was tested and verified on an 8GB Kingston SD card with *2013-02-09-wheezy-raspbian* as the host system but most Linux distributions should be able to work as a host platform for your LFS build as long as the required tools can be added. The minimum SD card size is probably 4GB. If you use the PiLFS base image, you're all set and you save a few hours on the total build time with more free space for your build compared to Raspbian.

Grand total build time is approximately 60 hours on RPi 1 and 12 hours on RPi 2.

1 The Big Choice

The first thing you have to decide before you start building is whether you're going to create an additional partition to hold your LFS system or use a PC to help finalize your SD card after a complete build.

The new partition method:
This is your only option if you don't have access to a PC with an SD card slot that can mount and modify an ext4 file system. You will add a new partition to the SD card, build your LFS system, then make the bootloader use your new LFS partition as the root filesystem.

The PC reshuffle method:
With this method you will build a complete LFS system inside a regular directory on the existing partition of the SD card. After completion, you'll move the SD card over to a PC, mount it, erase everything except your LFS directory, and finally move your LFS system into place to the root of the card. Thus replacing the old distribution with your own.

Partition resize?
When booting up a new distribution for the first time, you probably had the choice of resizing the partition to fit the space available on your SD card. With the two above methods in mind, you need to decide if you should resize your existing partition or not - if you're adding a new partition, you shouldn't. But if you're going with the PC reshuffle method, you should.

Perhaps you've already done the resizing a long time ago and now you're thinking about shrinking the partition to make room for a new one. In that case you're on your own... it's probably a lot easier to just re-image the card.

2 Preparing Your Distribution [2.2. Host System Requirements (http://www.linuxfromscratch.org/lfs/view/development/chapter02/hostreqs.html)]

It's a good idea to have the maximum amount of RAM available for building. Either use raspi-config to change your memory split to 16 or change it manually by adding `gpu_mem=16` to `/boot/config.txt` and reboot.

If you're building on a Pi with 256MB of RAM you'll need some swap space (beyond.html#addswap).

If you're on Raspbian, set a root password with `sudo passwd root` and login as root.

Now, if you run the `version-check.sh` script on Raspbian you'll see that there are four packages missing from the requirements. Let's add those:

```
apt-get update
apt-get install bison gawk m4 texinfo
```

3 Adding an LFS Partition [2.4. Creating a New Partition (http://www.linuxfromscratch.org/lfs/view/development/chapter02/creatingpartition.html)]

So if you've decided on the new partition method, use fdisk or cfdisk to do some work on your SD card's partition table:

```
cfdisk /dev/mmcblk0
```

mmcblk0p1 is the FAT32 partition that is mounted under */boot* and contains the bootloader and Linux kernel.

mmcblk0p2 contains the host system and is made to fit on a 2GB card in the case of Raspbian.

Hopefully you'll see some **Free Space** at the bottom of the table. Select it and make a **New Primary** partition with whatever space is left. You then need to **Write** the changes to disk before exiting. You may need to reboot at this point for Linux to see your new partition.

What about swap partitions?
There is no need to create a dedicated swap partition! A swap file (beyond.html#addswap) can be added at any time later on and is more flexible while offering the same performance.

4 Creating an ext4 File System [2.5. Creating a File System on the Partition (http://www.linuxfromscratch.org/lfs/view/development/chapter02/creatingfilesystem.html)]

Use the following command to create an ext4 file system on the new partition:

```
mkfs.ext4 -m 1 -L MyLFS /dev/mmcblk0p3
```

5 Setting the $LFS Variable [2.7. Mounting the New Partition (http://www.linuxfromscratch.org/lfs/view/development/chapter02/mounting.html)]

This is where your choice of build method comes into play again. If you're building on a new partition, do this:

```
export LFS=/mnt/lfs
mkdir -pv $LFS
mount -v -t ext4 /dev/mmcblk0p3 $LFS
```

If on the other hand you're building in a regular directory for the PC reshuffle method, you would just do this:

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

6 Grabbing All The Packages [3.1. Introduction (http://www.linuxfromscratch.org/lfs/view/development/chapter03/introduction.html)]

Since the Pi needs a slightly different set of packages and patches, we use our own PiLFS wget-list to grab everything (including the build scripts etc):

```
mkdir -v $LFS/sources
chmod -v a+wt $LFS/sources
cd $LFS/sources
wget http://intestinate.com/pilfs/scripts/wget-list
wget -i wget-list -P $LFS/sources
```

7 Adjusting the lfs User Environment [4.4. Setting Up the Environment (http://www.linuxfromscratch.org/lfs/view/development/chapter04/settingenvironment.html)]

Again, you need to change `LFS=/mnt/lfs` to `LFS=/lfs` if you are not building on a new partition.

Most importantly, the *LFS_TGT* variable needs to be changed to `LFS_TGT=$(uname -m)-lfs-linux-gnueabihf`.

8 Building Chapter 5 [5.4. Binutils - Pass 1 (http://www.linuxfromscratch.org/lfs/view/development/chapter05/binutils-pass1.html)]

Alright, this is the moment of truth. Will you make your way through chapter 5 slow and steady, or just execute the build script and go do something else for 30 hours? This is for you to decide ... I've done both :)

Also, here is where you might want to start a **tmux** or **screen** session. Because if you break your ssh connection or your router has a hickup, your build will just stop.

So you've made up your mind? Okey, script runners, start your engines:

```
cd $LFS/sources
chmod +x ch5-build.sh
./ch5-build.sh
```

The script will report your SBU time after the first binutils build has finished, mine is usually around 1 hour.

For those brave souls who are working through the chapter by hand, check out the about page and read the build scripts to figure out what's going on.

9 Building Chapter 6 [6.7. Linux API Headers (http://www.linuxfromscratch.org/lfs/view/development/chapter06/linux-headers.html)]

Cool, you've made it this far. You should have received the total build time for Chapter 5. Oddly enough, the build time for Chapter 6 is roughly the same amount of hours.

Before you enter the chroot and start your build, you'll want to edit the `ch6-build.sh` script to set a couple of optional parameters at the top. Then enter the chroot and execute the script:

```
cd /sources
chmod +x ch6-build.sh
./ch6-build.sh
```

10 Introducing the PiLFS Bootscripts [7.6. System V Bootscript Usage and Configuration (http://www.linuxfromscratch.org/lfs/view/development/chapter07/usage.html)]

As you may be aware the Pi doesn't have a way of keeping time between reboots. When it boots, it has no idea what time it is until it can fetch the correct time from an NTP server.

To address this issue and a few other Pi specific things, I've bundled a few bootscripts (beyond.html#pilfsbs) together from different sources into a tarball for easy installation. I recommend that you add these four scripts:

```
make install-networkfix install-swapfix install-fake-hwclock install-switch-cpu-governo
```

11 Creating the fstab [8.2. Creating the /etc/fstab File (http://www.linuxfromscratch.org/lfs/view/development/chapter08/fstab.html)]

First up is an fstab suitable for the new partition method. Here we take the old distribution's partition and turn it into a dedicated space for the user's home directories (you'll have to erase the old content after the first boot obviously).

```
/dev/mmcblk0p1 /boot        vfat     defaults             0     0
/dev/mmcblk0p2 /home        ext4     defaults,noatime     0     1
/dev/mmcblk0p3 /            ext4     defaults,noatime     0     2
#/swapfile     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
```

If you're going with the PC reshuffle method, your fstab would instead look something like this:

```
/dev/mmcblk0p1 /boot        vfat     defaults             0     0
/dev/mmcblk0p2 /            ext4     defaults,noatime     0     1
#/swapfile     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
```

12 Kernel and Bootloader [8.3. Linux (http://www.linuxfromscratch.org/lfs/view/development/chapter08/kernel.html)]

While you could potentially build (beyond.html#kernelbuild) your own Linux kernel as a part of your LFS build, I would strongly recommend that you ensure your system boots properly with the Raspberry Pi Foundation's kernel first.

There is no GRUB on the Pi, and we rely instead on the Raspberry Pi Foundation's bootloader to boot us up.

If you added a new LFS partition, exit the chroot at this point and edit `/boot/cmdline.txt`. You need to change `root=/dev/mmcblk0p2` to `root=/dev/mmcblk0p3` so that the bootloader will use your new LFS partition as the root filesystem.

13 The End [9.3. Rebooting the System (http://www.linuxfromscratch.org/lfs/view/development/chapter09/reboot.html)]

So there you have it. Your system is now bootable. If you're like me, you'll want to add a couple of things before you take the plunge though. I would at least add dhcpcd (http://www.linuxfromscratch.org/blfs/view/svn/basicnet/dhcpcd.html), wget (http://www.linuxfromscratch.org/blfs/view/svn/basicnet/wget.html), OpenSSH (http://www.linuxfromscratch.org/blfs/view/svn/postlfs/openssh.html) and ntp (http://www.linuxfromscratch.org/blfs/view/svn/basicnet/ntp.html) so that you can easily start building more packages from Beyond LFS/PiLFS once inside your new shiny LFS system.

14 The PC Reshuffle Process

Now I'll take you through the process of fixing up your SD card on a Linux PC.

On my netbook, the SD card shows up as `/dev/sdb`, so I mount the data partition like so:

```
mount /dev/sdb2 /mnt
```

Then I erase everything except the `/lfs` directory, like so:

```
cd /mnt
shopt -s extglob
rm -rf !(lfs)
```

And now we do the old switcheroo:

```
mv /mnt/lfs/* /mnt
```

The mv command will preserve all the file permissions by default, but if you'd like to copy your LFS system around instead, don't forget to add the preserve permissions flag to cp, i.e. `cp -rp`.

15 Bonus: How To Make an SD Card Image Suitable For Distribution

Perhaps you too would like to share your SD card image with friends or the internet at large? The tricky part of preparing an SD card image is that a straight image copy using dd would take the same size as your whole SD card. Here's how to prepare an SD card image from the ground up:

First, make sure you have a complete copy of all the files that are going to go on the card, including the stuff that goes onto the FAT32 partition like the bootloader and kernel.

Next, we're going to recreate the partition table from scratch. I prefer to use Parted (http://www.linuxfromscratch.org/blfs/view/svn/postlfs/parted.html) for partition editing because of its automatic alignment feature. Erase whatever is on your card, then create a nice layout:

Don't edit the wrong disk ... triple-check! :)

```
parted /dev/sdb
mkpart primary fat32 1 50
```

This creates the FAT32 boot partition, from the first available sector, 50MB in size. Now if you print the table, you should have something like this:

```
Number  Start   End     Size    Type     File system  Flags
 1      1049kB  50.3MB  49.3MB  primary  fat32        lba
```

Notice how the partition ends at 50.3MB - this will be the start for our data partition:

```
mkpart primary ext4 50.3 950
```

This creates our ext4 data partition, beginning after 50.3MB and ending at 950MB making the partition 900MB in size.

Here we're aiming for an image that could fit snuggly on a 1GB SD card but this would probably require an LFS system with all binaries stripped to fit. Feel free to adjust for 2GB.

Now that we have our partition layout, let's format the partitions, like so:

```
mkdosfs -F 32 -n Pi-Boot -v /dev/sdb1
mkfs.ext4 -m 1 -L MyLFS /dev/sdb2
```

Then copy all the binaries into place:

```
mount /dev/sdb1 /mnt
cp -rvp bootfiles/* /mnt
umount /mnt
mount /dev/sdb2 /mnt
cp -rvp lfsfiles/* /mnt
umount /mnt
```

Alright, with our card prepared, this is where the magic happens. We're using dd to make an image of the card but we want to stop reading exactly where our ext4 partition ends:

```
dd if=/dev/sdb of=mylfs.img bs=1M count=950
```

So you'll want to set the `count` to wherever your ext4 partition ends. That's all there is to it.

Build Scripts

- wget-list (scripts/wget-list)
- ch5-build.sh (scripts/ch5-build.sh)
- ch6-build.sh (scripts/ch6-build.sh)
- pilfs-bootscripts-20181124.tar.xz (scripts/pilfs-bootscripts-20181124.tar.xz)